Sniffer™

Network and Protocol Reference

Network General

DISCLAIMER OF WARRANTIES

The information in this document has been reviewed and is believed to be reliable; nevertheless, Network General Corporation makes no warranties, either expressed or implied, with respect to this manual or with respect to the software and hardware described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. The entire risk as to its quality and performance is with the buyer. The software herein is transferred "AS IS."

Network General Corporation reserves the right to make changes to any products described herein to improve their function or design.

In no event will Network General Corporation be liable for direct, indirect, incidental or consequential damages at law or in equity resulting from any defect in the software, even if Network General Corporation has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

Sniffer is a trademark of Network General Corporation.
All other registered and unregistered trademarks in this document are the sole property of their respective companies.

Document prepared by David M. Trousdale
May 1990

P/N 20026-001

Table of Contents

List of Figures

Preface

Network General

Preface

The *Network and Protocols Reference* provides background information on networks and protocols. You will want to refer to it from time to time to help you get the most out of your Sniffer™ network analyzer.

Three chapters cover basic knowledge about local and wide area network architectures, network protocols, and protocol interpreters. Chapter 1 explains some basic differences and similarities between the network architectures to which you can attach an analyzer. In Chapter 2, you will find information about the various protocol suites interpreted by the Sniffer analyzer. Finally, Chapter 3 details how you can construct your own custom protocol interpreter or extend an existing one.

1. Network Architectures

1

Network General

Chapter 1. Network Architectures

Chapter 1. Network Architectures

Chapter Overview

The network reference material in Chapter 1 covers the physical and data link layers of each of the networks supported by the Sniffer network analyzer. Each section summarizes the features of each particular network and the variations related to the use and understanding of the Sniffer analyzer.

Ethernet® Network Architecture

Ethernet is a type of local area network (LAN) suitable for high-speed interconnection of computers and computer-controlled devices over moderate distances. The architecture of Ethernet is defined de facto by implementations from many manufacturers and de jure as ANSI/IEEE standard 802.3, ISO/DIS standard 8802/3, and FIPS standard 107.

There are several variations permitted by the standards and other variations that are not official but are, nonetheless, popular. So to say that a network is an "Ethernet" means only that it is one of several closely-related, but not necessarily compatible, LANs. Even use of the word "Ethernet" itself may be confusing, since it is sometimes reserved to refer to the earlier DEC/Intel/Xerox (DIX) network, leaving 802.3 networks to be used for the others.

Some system implementations of the Ethernet network also use at least a subset of a similarly standardized protocol for Logical Link Control, referred to as LLC and defined as ANSI/IEEE standard 802.2 and ISO/DIS standard 8802/2.

Physical Interconnection and Speed

Stations connected to a conventional Ethernet network are all connected to the same bus, so that every station hears what any station transmits. The delay between transmission and reception depends only on the propagation delays through the wires and attaching devices. In this way Ethernet is similar to networks like StarLAN and IBM PC Network (see "StarLAN Network Architecture" and "IBM PC Network Architecture"). On the other hand, it differs fundamentally from networks like the IEEE 802.5 token ring, where stations are wired in a *logical ring* so that each station only hears what its upstream neighbor transmits (see "Token Ring Network Architecture").

The most common Ethernet transmission speed is 10 megabits per second (Mbps), or 1,250,000 bytes per second, sent in baseband (non-modulated, non-RF) form.[1] The ANSI/IEEE documents refer to this as the *10BASE5* (10 Mbps, baseband, and 500 meters per segment) standard. Another common standard is *1BASE5* (1 Mbps, baseband, and 500 meters per segment) over twisted-pair cable that is based on AT&T's 1 Mbps StarLAN. A third standard is 10BASE2 (10 Mbps, baseband, and

---

1 Because of the access control mechanism described below, the effective network throughput is significantly less than the transmission speed.

185 meters per segment). A newer, but increasingly popular, standard is *10BASE-T* (10 Mbps, baseband, over twisted pair). The Ethernet Sniffer supports 10BASE5 networks, 10BASE2 networks, and some variants of them. There is a separate StarLAN Sniffer analyzer for 1BASE5 networks.

Thick Ethernet

There are two common schemes for the wiring of stations into an Ethernet. The original *thick Ethernet* scheme uses a backbone of yellow semi-rigid 0.4" diameter coaxial cable segments for up to 100 stations per segment. Each segment is a maximum of 500 meters long, and segments may be connected with repeaters subject to the restrictions that there are no more than 2 repeaters in the path between any two stations. Thus the maximum total cable length between two stations is 1500 meters but is often less than that in practice because of the way the cable is routed. The cable impedance is 50 ohms, and segments must be terminated on each end with a 50-ohm terminator attached with an N-series coaxial connector. The shield conductor must be grounded to an earth reference at only one point and fully insulated everywhere, including at connectors and terminators, to prevent shock hazards.

A station is connected to a thick Ethernet cable by means of a *transceiver*, which is a signal converter that must be attached directly to the cable. (Transceivers are called *medium attachment units* (MAU) by the IEEE Standards documents.) Many transceivers clamp on to the cable and use a *vampire tap* to make contact with the outer shield and inner conductor without requiring that the cable be cut. The spacing between transceivers must be a multiple of 2.5 meters; the yellow cable has black marks to indicate possible transceiver attachment points.

A single station is connected to its transceiver with a more flexible 4-pair *drop cable* or *Attachment Unit Interface* (AUI) *cable* whose maximum length is 50 meters. In addition to carrying network data in each direction on separate pairs, the drop cable also supplies power for the transceiver electronics from the equipment being attached to the network. Both ends of the drop cable use DB-15 connectors: a male connector with locking posts for the equipment end, and a female connector with a slide latch for the transceiver end. Note that the slide latch is often too big for the back panel of personal computers, so at least two variations of this attachment scheme are in common use.

To reduce the per-station cost of the transceiver and to circumvent the limit of 100 stations per segment, transceiver fan-out boxes are also available. These typically allow four or eight drop cables to be connected to a single box, which in turn is connected to a standard transceiver clamped to the coaxial cable segment.

Thin Ethernet

Another approach to reducing the cost of an Ethernet installation is *thin Ethernet*, "cheapernet," or 10BASE2. It dispenses with the large, semi-rigid coaxial cable and separate external transceivers. Instead, it uses flexible RG-58/U coaxial cable and transceiver circuitry built into each attaching computer. For this wiring scheme, the overall topology is still a linear segment, but now the segment passes by the back of each computer. Attachment is made by splicing BNC connectors into the cable and inserting a "T" connector. The segments are still terminated with 50-ohm terminators at each end. The maximum length of a segment is usually 185 meters, although some vendors support longer distances.

Cheapernet transceivers are generally part of the network adapter circuitry, and the exposed connector is, thus, a female BNC. A hybrid approach is also possible, since there are *thin Ethernet transceivers* available that attach with a standard drop cable to an attaching computer but have a female BNC instead of a *vampire tap* for connecting to the network. There are also *thin to thick adapters* that allow segments to be built out of both kinds of cable. Finally, some devices—like the Sniffer analyzer—provide both a BNC connector (for thin Ethernet) and a DB-15 (AUI) connector (for a thin- or thick-Ethernet transceiver). On the Sniffer analyzer, you determine which connector is active by the position of a jumper or the board's setup software.

Twisted Pair Ethernets

Ethernet over twisted pair has been standardized by an IEEE 802.3 project. It is known as the 10BASE-T standard, and the standard provides a means for attaching AUI-compatible devices to 24 gauge, unshielded twisted pair cable, instead of the usual coaxial media.

Stations connect their standard AUI connector to a special twisted-pair transceiver (confusingly called a *Twisted Pair MAU* by the IEEE Standard documents) that has an RJ-45 phone jack on the other side. The transceiver is then wired to one port of a *multiport repeater* (MPR) typically located in a wiring closet. The MPRs can in turn be wired to higher-level MPRs. The resulting network topology is, thus, a distributed star, in which each twisted-pair cable is allowed to be up to 100 meters long. Note that this is physically very unlike the original multidrop Ethernet but can be centrally managed more easily and arguably has higher reliability.

There are two earlier twisted pair Ethernet architectures: AT&T's StarLAN 10 and LattisNet from SynOptics Communications. These are likely to decrease in popularity as the 10BASE-T standard dominates, but most computers (and network analyzers) with an AUI interface can be used interchangeably on any of these networks with the appropriate transceiver.

Other Ethernets

Ethernet variations proliferate both with and without benefit of official standardization. Most share at least a philosophical tradition with the Ethernets discussed above but are often incompatible in the sense that equipment designed for one species cannot be connected with equipment designed for another. Prominent in this menagerie are:

- Broadband Ethernets, some of which run at 5 Mbps so as to fit within a single television channel assignment (see the section, "PC Network Architecture").
- Fiber-optic Ethernets, most of which use two cables per station and a centralized optical coupler to rebroadcast the transmitted signal to all receivers.

Access Control

Ethernet and its variants, StarLAN and PC Network, use an algorithm to control transmission called *carrier sense multiple access with collision detection* (CSMA/CD). A similar architecture is *carrier sense multiple access with collision avoidance* (CSMA/CA). Apple Computer implemented CSMA/CA in its LocalTalk® network

using the LocalTalk Link Access Protocol or LLAP (see the section, "LocalTalk Network Architecture").

Before transmitting, a station waits until it hears no other station transmitting. It defers until it senses no carrier from another station and then sends its message. Since there is the possibility that another station may decide to transmit at roughly the same time, the sending station monitors the network to hear if its own message appears on the network ungarbled.[1] If it detects a collision with another message, it intentionally sends a few additional bytes (a process called *jamming*) to ensure propagation of the collision throughout the system to all other transmitting stations. The station then remains silent for a randomly-determined amount of time (controlled by a *backoff algorithm*) before attempting to transmit again.

Collisions may be heard by receiving stations as badly-formatted frame fragments called *runts* that are typically shorter than the minimum frame size[2] and have incorrect check words. These frames may also have an incorrect starting sequence and may not be seen as the start of a frame at all. Note that the propagation time through Ethernet and StarLAN networks is typically much longer than the transmission time for a bit. Therefore, different receivers will see different effects depending on their position relative to the various transmitters (and, for StarLAN, to hubs).

In addition to deferring to active traffic, stations wait before beginning transmission after the end of another transmission. The pause varies according to network type: 9.6 microseconds for Ethernet, 96 microseconds for StarLAN, and 48 microseconds for PC Network. This enforced interframe spacing allows time for receivers to recover from one frame and to prepare to receive the next.

Other Transceiver Functions

Besides moving serial data to and from the attached device and the network, the transceiver (or MAU) has several other functions:

- ***Collision detection:*** One wire pair in the drop cable transmits the *collision detect signal* (called *signal quality error*, SQE, in IEEE documents) from the transceiver to the device. In addition to its use to control transmission, this signal can also be used to detect some network and transceiver failures. A transceiver that supports the *signal quality error test* feature will generate a collision detect signal at the end of a transmitted frame to demonstrate that the collision detect circuitry is at least partially operational.

- ***Jabber detection:*** The transceiver is required to disable transmission if the device transmits for longer than the longest possible frame. This prevents some kinds of device failures from halting network activity but is not foolproof.

---

1 For PC Network, the sending station monitors the network on the frequency to which the headend translates the traffic.

2 PC Network permits a smaller frame size than Ethernet and StarLAN. A PC Network data field may be as small as four bytes.

Format of a Frame

All data in a frame is transmitted as a sequence of 8-bit bytes starting with the least-significant bit. The format of each frame is as follows:

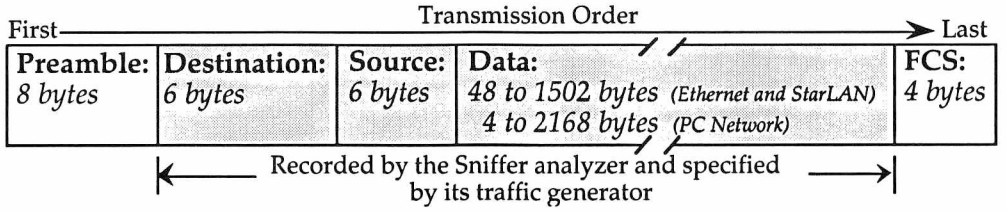

Figure 1. Frame format for Ethernet, StarLAN, and PC Network.

The **preamble** is a fixed data pattern used for receiver synchronization and recognition of the start of a frame. The **destination** and **source** addresses are each 6 bytes. Various kinds of multicast addresses are indicated if the first transmitted bit (the low-order bit of the first byte) of the destination address is a one. Note that the IEEE 802.3 standard also permits 2-byte source and destination addresses, but their use is rare and inconsistent with 10BASE5 and 1BASE5.

If the first transmitted bit of the source address is a one, it indicates that a *routing information* (RI) field is present before the data field. This is a convention inherited from token ring. Such frames are generally seen on an Ethernet only when forwarded over a MAC-level bridge from a token ring.

The **data** field must have a minimum number of bytes so that the duration of a frame is longer than the worst-case propagation delay[1] through the network. For Ethernet and StarLAN, the minimum must be 48 bytes so that the frame is at least 60 bytes (excluding the **preamble** and the **FCS**). For PC Network, the minimum must be 4 bytes so that the frame is at least 28 bytes. If this were not true, then collisions might not be detected.[2]

Following the data is a 4-byte *frame check sequence* (**FCS**) that checks the validity of the source, destination, and data fields. The Sniffer analyzer does not record the **FCS** but will optionally collect and identify frames whose **FCS** is incorrect.

Format of the Data

To specify how the initial bytes of the data field are to be interpreted, there are several conventions in general use. The original format as defined by Xerox®, now often called the "Ethernet" format (in contrast to the IEEE 802.3 format), contains no length field and begins with a 2-byte **Ethertype** field that indicates the major protocol type. For example, the IP protocol of TCP/IP is assigned the Ethertype hex 0800.

---

1 45 microseconds for Ethernet.

2 The IEEE 802.3 standard specifies other limits on the frame size for non-10BASE5 and non-1BASE5 networks only.

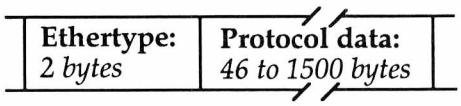

Figure 2. Original "Ethernet" data format defined by Xerox.

The assignment of Ethertypes to protocols was originally done by Xerox and has now been taken over by the US Defense Communications Agency.

The second convention for interpretation of the data field is that supported by the IEEE/ANSI/ISO standards organizations for 802.3 networks and begins with a 2-byte **length** field (most significant byte first), followed by a *Logical Link Control* (LLC) header that conforms to the IEEE 802.2 standard:

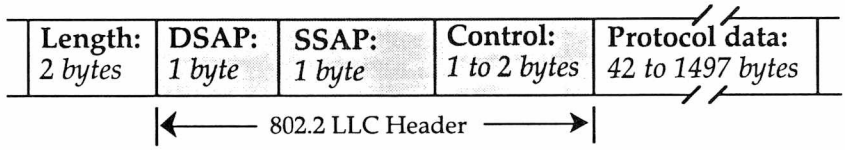

Figure 3. IEEE 802.3 data format defined by IEEE/ANSI/ISO standards organizations.

Note that although both data formats can and do appear on the same network, there is, in theory, no foolproof way to distinguish between the two by looking at the frame. In practice, though, it turns out that almost all assigned Ethertypes are numerically larger than the maximum frame length of 1500. Thus, if the first two bytes of frame data are larger than 1500, it is probably an Ethernet/Ethertype frame. (The only common exception is the PUP Ethertype, hex 0200.)

The Sniffer analyzer decodes frames in either format and makes the format decision automatically unless instructed otherwise by the operator.

For frames of either type, the embedded **protocol data** represents information that is interpreted by higher level protocols and may contain many other such nested levels.

A third convention, the original format used by IBM® and Sytek, applies only to PC Network. It begins with a 2-byte **length** field that is then immediately followed by the **NetBIOS header.** There is no identification of the protocol type, so this format cannot easily be distinguished from other protocols on the network. Note that this is not the same NetBIOS format as is used on token ring networks or on PC Network using the IBM LAN Support Program.[1]

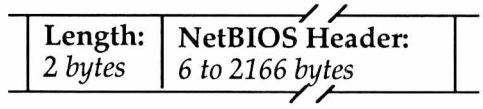

Figure 4. The original data format used by IBM and Sytek. It applies only to PC Network.

1 As of this writing, the Sniffer analyzer does not decode the original PC Network NetBIOS format frames, described in the following section.

LLC Frames

A frame that conforms to the 802.2 standard is an LLC frame. LLC is a protocol that provides reliable connection-oriented virtual circuits or connectionless datagrams between processes. It is a subset of the International Standards Organization-defined superset, *High-level Data Link Control* (HDLC). See the subsection on HDLC and two other HDLC subsets, SDLC and LAPB, in the section, "WAN/Synchronous Architecture."

The **protocol data** part of LLC frames (see Figure 3) begins with a 3 or 4 byte header, the first two bytes of which are the *destination service access point* (**DSAP**) and *source service access point* (**SSAP**). The SAP numbers are preassigned codes that indicate which sub-protocol is used in the rest of the frame. For example, the NetBIOS protocol[1] has been allocated a single SAP (hex F0), and Systems Network Architecture (SNA) has been allocated four SAPs (hex 04, 05, 08, and 0C). The **SSAP** often equals the **DSAP**, except in frames that are establishing an initial SNA connection.

The **control** field defines the exact function of LLC frames. There are three LLC frame types:

Information I format frames are used to send arbitrary sequenced data interpreted by the protocol that the SAPs designate. The LLC header contains a *send* sequence number N(S) for this frame and a *receive* sequence number N(R) of the next frame expected from the other station.

Supervisory S format frames contain or consume an N(S) number but do contain an N(R) number. In addition they can contain the following indications as either a command or a response:

RR	*Receive Ready*. Transmission of Information frames can proceed.
RNR	*Receive Not Ready*. Transmission is temporarily blocked.
REJ	*Reject*. Retransmission starting with N(R) number is requested.

Figure 5. Indications used in LLC supervisory frames.

Unnumbered U format frames contain neither N(S) nor N(R) numbers but may contain control information or data. The commands and responses in the unnumbered frame format are:

1 This technique is also used on PC Network by the LAN Support Program.

SABME	*Set Asynchronous Balanced Mode (Extended).* Establish a virtual connection, also called a link.
DISC	*Disconnect.* Terminate a virtual connection.
DM	*Disconnected Mode.* The connection is broken.
UA	*Unnumbered Acknowledgement.* For SABME and DISC response.
FRMR	*FrameReject.* The format of a received frame was invalid. The **protocol data** field contains the reason.
XID	*Exchange Identification.* Used between two stations to exchange identification and the characteristics of the two stations.
TEST	*Test Probe.* Should be echoed by the receiving station.
UI	*Unnumbered Information.* Used by the SAP protocol for any purpose.

Figure 6. Commands and responses in LLC unnumbered frames.

The only LLC frames that are allowed to contain data after the LLC header are I, UI, TEST, XID, and FRMR types.

There are three types of LLC operation. The first is known as *unacknowledged connectionless service.* This is the minimal use of LLC in which every frame is a UI type. Thus, the data part of every frame viewed on a Sniffer analyzer begins with three bytes: the two SAP bytes and a UI (hex 03) control byte. This technique is typically used to implement various higher level protocols that do connection control and sequencing themselves, that do not need the services of the standard LLC, but that prefer compatibility with LLC formats.

The other two types carry more overhead. *Connection-oriented service* provides flow control, sequencing, and error recovery. *Acknowledged connectionless service* is also connectionless like the first type but provides for acknowledgement and relieves higher layers of that responsibility.

Assignment of Network Addresses

The modern convention for 6-byte node addresses is to divide them into an initial 3-byte code representing the manufacturer of the equipment, and a unique 3-byte serial or sequence number assigned to that piece of equipment. The responsibility to assign manufacturer codes was initially taken by Xerox but has now been assumed by the IEEE.

It is the intention of the IEEE that the same code be used for all networks supported by a manufacturer, including Ethernet/StarLAN (IEEE 802.3), PC Network, and token ring (IEEE 802.5). However, the fact that Ethernet/StarLAN/PC Network bytes are transmitted with the least-significant bit first, and token ring bytes are transmitted with the most-significant bit first has led to considerable confusion in the way that assigned addresses are used.

The IEEE's clearly stated position is that the assigned code specifies how the address should appear on the network cable. The result is that a network address stored in a computer's memory will be different, depending on what the originating network was. Since network addresses appear at various places within

the protocol levels of a frame, and since a frame may have been forwarded by various network types, this imposes a difficult burden on network software. In fact, observation of network software and hardware from a variety of vendors shows that some have chosen to use the same address as it appears in memory on both networks, thus using a code on one or the other network that, according to the IEEE interpretation, is not assigned to them.

Sytek/IBM 6-Byte Address

To add to the confusion, the original Sytek/IBM PC Network adapters used an entirely different convention for node addresses: 4 bytes of unique serial, followed by a 2-byte manufacturer's code. The only codes ever assigned were 0000 (for IBM) and 0100 (for Sytek). Both companies have since switched to the standard address format, but there are many currently-active network nodes using the old address format.

Token Ring Network Architecture

The token ring is a type of local area network suitable for high speed interconnection of computers and computer-controlled devices over moderate distances. The architecture of the token ring is defined de facto by implementations from IBM, Texas Instruments, and others and de jure as ANSI/IEEE standard 802.5 and ISO/DIS standard 8802/5.

Most system implementations of the token ring network also use at least a subset of a similarly standardized protocol for Logical Link Control, referred to as LLC and defined as ANSI/IEEE standard 802.2 and ISO/DIS standard 8802/2.

Physical Interconnection and Speed

Stations connected to the token ring network are wired together physically in star-like fashion. Each station uses one cable to attach itself to a nearby *passive concentrator*, or *multiple access unit* (MAU).[1] The MAUs can themselves be linked together and may be separated by moderate distances. The number of stations and the distance limitations depend on several variables, including cable type, but typically one or two hundred stations can be interconnected into a single network segment using cables between stations and MAUs up to 300 meters long. The distance limitations can be overcome by using special line drivers or fiber optic cables. Networks of many hundreds or thousands of stations over large distances can be created using bridges between network segments.

One type of connector used to attach to MAUs was designed by IBM and called the IBM Data Connector. They are hermaphroditic, so that any two may be joined; cable "extension cords" have the same connector on both ends. The cable that connects to a particular computer may use the hermaphroditic connector if there is enough panel space for the mating connector (about one inch by one inch) or may use a non-standard connector. The convention for personal computers is to use a DB-9 female connector on the backpanel and DB-9 male on a cable whose other end has the hermaphroditic connector. Other vendors have built MAUs with RJ-11 connectors that use less panel space

There are two basic speeds for the network: 4 Mbps, or 500,000 bytes per second, and 16 Mbps, or 2,000,000 bytes per second. This does not include the many levels of overhead in a typical application, and throughput for the user will often be many times less than that. There is nothing in the hardware or software architecture that limits the network to 16 Mbps.

Stations operating at 4 and 16 Mbps may not be attached to a token ring at the same time. The first station to become operational on the ring establishes the speed, and any subsequent station entering the ring at a different speed will cause transmission errors and will often result in other stations removing themselves from the ring. Unfortunately, current ring adapters have no provision for determining the speed of an established ring before attempting to join it.

---

1 Do not confuse this use of the MAU acronym with *Medium Attachment Unit* (MAU) defined in IEEE Standards documents for Ethernet.

Logical Interconnection

Each interconnection cable contains two twisted pairs of wires. Although the stations and MAUs are cabled in a star-like fashion, the electrical effect of the token ring cables and connectors is to create a continuous ring from station to station. One twisted pair in the cable to each station is used to transmit to the next station in the ring, and the other pair is used to receive from the previous station. The ordering depends on how cables are plugged into the MAUs and how the MAUs are interconnected.

The operation of the ring depends on each station retransmitting data from its *receive pair* onto its *transmit pair*, regardless of whether that station is involved in the conversation. To insure that the ring is operational even when some stations are turned off, connecting a cable from a station to an MAU is not sufficient to cause that station to enter the ring; it also must send a DC voltage on its transmit pair to trigger a relay in the MAU. If power to the station fails, or if the cable to the station is disconnected at either end, the relay loses power and the ring bypasses that station.

When the relay is not powered, the cable to the station has its transmit pair connected to its own receive pair so that it may test the network adapter and cable. Prior to inserting itself onto the ring by supplying the relay voltage, the network adapter sends several thousand data frames to itself to verify correct operation. This process, plus the network adapter self-test, may take 15 seconds or more.

Access Control

Only one station on the entire ring is allowed to transmit data at a time. To control access, a 3-byte message giving permission to transmit, called the *free token*, continually circulates when there is no other traffic. Each idle station retransmits it as it is received. A station that wishes to transmit data waits for the token and then sends its data instead of the token. When its data transmission is finished, it regenerates the token message. In addition to this simple rotational priority scheme, there are also ways to establish other priorities. Every message (including the token) contains both a 3-bit priority field for itself and a 3-bit reservation priority for a possible subsequent message.

A data message, called a *frame*, may be directed to a single destination station or to any of various groups of stations. In all cases, the addressee (the station that receives the message) does *not* remove it from the ring. It simply makes a local copy of the message and retransmits it. The originating station is responsible for removing the message from the ring when the message returns to it. The originator then replaces the message with the token.

In visualizing the traffic flow on the network, it is important to realize that most frames are much longer (in time) than the round-trip delay around the ring, particularly at 4 Mbps. Each station introduces a delay of less than 3 bit times when it is repeating data from its receive cable to its transmit cable, whether or not it is making a copy of the data. That delay for each station, plus the cable propagation delays, produce the total ring round-trip time. For a typical network of 50 stations, the round-trip time might be about 50 microseconds. A 1000 byte (8000 bit) frame takes 2000 microseconds to transmit at 4 Mbps, so the transmitter

must be removing the beginning of the frame that has made the trip around the ring long before it has finished sending out the whole frame.

At 16 Mbps, small frames will take less time to transmit than the round-trip token timing, especially for large networks. To improve performance, the newer token ring adapters implement an *early token release algorithm* that allows them to transmit the free token to the next station before they have received a frame just transmitted.

In normal operation, the token is circulated and regenerated by the cooperative operation of all stations acting democratically. If the token is destroyed by transmission error or other fault (a station attaching or removing from the ring typically destroys the token because of electrical noise created by the relay operation) it is the responsibility of a station designated as the *active monitor* to notice the absence of the token and to regenerate it. There is only one active monitor on the network at a time, although every station is able to assume the role if needed.[1] If the active monitor is disabled or leaves the ring, a monitor contention process begins through which a new active monitor is elected by the remaining stations.

Format of a Token Frame

All data is transmitted as a sequence of 8-bit bytes sent serially and Manchester encoded. The minimum transmission is the 3-byte *token*.

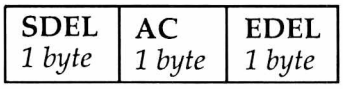

Figure 7. Frame format for a token frame.

Both *starting delimiter* (**SDEL**) and *ending delimiter* (**EDEL**) have intentional Manchester code violations in certain bit positions so that the start and end of a frame can never be accidentally recognized in the middle of other data. The *access control* (**AC**) byte contains a bit that indicates that this is a token, not a data frame, and contains priority information. Tokens are not recorded by the Sniffer analyzer.

Format of a Data Frame

If a message is not a token, then it is a *data frame*.

[1] The Sniffer analyzer cannot play the role of active monitor when in capture mode, however.

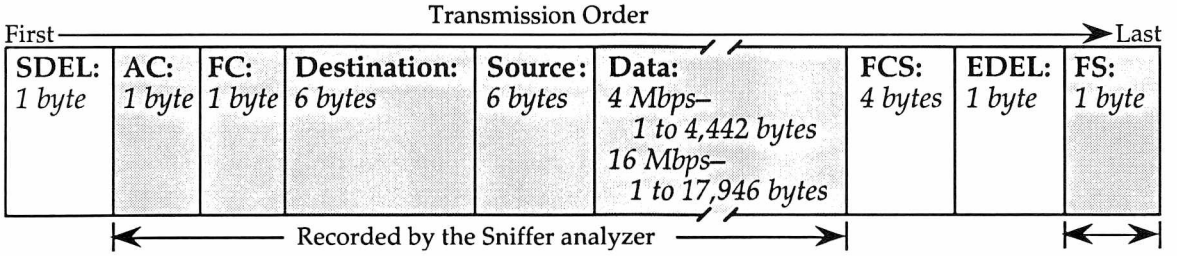

Figure 8. Frame format for a token ring data frame.

The **SDEL**, **AC**, and **EDEL** fields are as before, except the the **AC** byte now says that this is a data frame and not a token. The **FC** byte contains frame information. The **destination** and **source** addresses are each 6 bytes, and various kinds of *multicast* or *broadcast* addresses are indicated if the first bit of the destination address is a one. Following the **data** field is a *frame check sequence* (**FCS**) that checks the validity of all previous data starting with the **AC** byte. The Sniffer analyzer records bytes starting with **AC** and ending with the last byte of the data field.

The last byte for data frames is the *frame status* (**FS**) byte containing bits that may be set on by the recipient of the frame: *address recognized*, if a station matched the destination address and *frame copied* if it was able to successfully make a local copy of the data as it passed by. Note that the **FS** is not covered by the frame check sequence (so that the **FCS** doesn't have to be changed by the recipient), but for greater reliability, the bits in the **FS** are each duplicated. The Sniffer analyzer records the **FS** byte and displays it in the **detail** view.

Optional Routing Information

Token ring uses a technique known as *source routing* when it does routing. Thus, there is an optional *routing information* (**RI**) field that may be present at the beginning of the data part of any frame. The **RI** field, which may be up to 32 bytes long, contains information about the path that the frame took if it was forwarded through multiple network segments by bridges. If the **RI** field is present, the first bit of the source address will be a one.

The **RI** field is not part of the IEEE 802.5 token ring standard, although it has been proposed for official adoption. IBM software currently uses this extension to the standard.

MAC Frames

A data frame may be a *medium access control* (MAC) frame that contains information used to control the token ring network itself. Most MAC frames are generated and processed by the computer's network adapter and are not of concern to software within the host. The type field in the FC byte indicates whether the frame is a MAC frame.

MAC frames are used for processes like *monitor contention*, *error reporting*, and *error recovery*. In addition, the *active monitor* announces its presence with a periodic MAC frame, and all stations that could become the monitor (e.g., *standby monitors*), should the need arise, do likewise.

MAC frames contain a major type code followed by a variable number of variable-length fields called *subvectors* that give additional information.

LLC Frames

A data frame that is not a MAC frame is (in all non-proprietary uses of the token ring network) an LLC frame. See the discussion of the LLC frame format in the section, "Ethernet Network Architecture."

Since all non-MAC frames are supposed to use 802.2 LLC, an alternative mechanism has been standardized for encapsulating the older Ethertype formats. It is known as the Sub-Network Access Protocol (SNAP), and you can see the frame format in Figure 9.

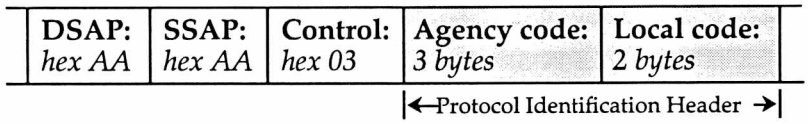

Figure 9. LLC frame format for Sub-Network Access Protocol (SNAP).

SNAP has been assigned **SAP** hex AA, and the data field following the **Control** field is a five-byte *protocol identification header*. The **Agency code** is an assigned manufacturer code, but sometime 0 is used. The **Local code** is typically the Ethertype. Ethernet-style data followed the header.

Assignment of Network Addresses

A discussion of the assignment of network addresses can be found in the section, "Ethernet Network Architecture."

StarLAN™ Network Architecture

The original 1 Mbps StarLAN is a type of LAN suitable for moderate-speed interconnection of computers and computer-controlled devices over moderate distances. The architecture of StarLAN is defined de facto by implementations from several manufacturers and de jure as a proposed variation of ANSI/IEEE standard 802.3, ISO/DIS standard 8802/3, and FIPS standard 107. StarLAN is, thus, a slower speed variant of the network popularly called Ethernet (see the section, "Ethernet Network Architecture").

Physical Interconnection and Speed

Stations connected to a StarLAN network are connected so that every station hears what any station transmits. The delay between transmission and reception depends only on the propagation delays through the wires and attaching devices. In this way StarLAN is similar to networks like Ethernet and IBM PC Network (see the sections "StarLAN Network Architecture" and "IBM PC Network Architecture"). On the other hand, it differs fundamentally from networks like the IEEE 802.5 token ring, where stations are wired in a logical ring so that each station only hears what its upstream neighbor transmits (see "Token Ring Network Architecture").

The original StarLAN transmission speed is 1 Mbps, or 125,000 bytes per second, sent in baseband (non-modulated, non-RF) form. The ANSI/IEEE documents refer to this as the *1BASE5* (1 Mbps, baseband, and 500 meters per segment) standard. Another common standard is *10BASE5* (10 Mbps, baseband, and 500 meters per segment), for Ethernet. Because of the access control mechanism described below, the effective network throughput is significantly less than the transmission speed. StarLAN 10 is a newer version, and it operates at 10 Mbps.

Although the StarLAN network is logically a bus because every station hears what every other station transmits, it is physically wired as a hierarchical branching star. Each station is wired to a repeater unit variously called a *network extension unit* or *hub*. Each hub can accept connections from 7 to 11 stations or other hubs, and also has an output signal that is used to connect to a hub at the next higher level in the hierarchy, if any. Hubs can be layered up to five deep. The top hub being the *header* or *master* hub of the network.

The cable used to connect stations to hubs and between hubs contains two twisted pairs of voice-grade 24 AWG or heavier wire. One pair is for transmitted data and one for received data. The standard connectors are four-pair RJ-45 telephone plugs and RJ-18 jacks with two of the pairs unused. Note that these are larger than the three-pair connectors used for most phones.

The maximum distance between a station and the hub, or between hubs, is 250 meters. Thus the maximum cable distance from any station to the master hub is five times that, or 1250 meters. The maximum number of stations in any single network is 1024.

Some manufacturers specify or recommend limits that are more restrictive than these. In addition, some permit up to 10 stations to be connected together in a

chain without any hubs if the maximum cable distance of the chain is less than 125 meters.

Access Control

StarLAN networks, like Ethernet and PC Network, use the CSMA/CD algorithm to control transmission. See the discussion of access control in the section, "Ethernet Network Architecture."

Format of a Frame

The frame format used in StarLAN networks is identical to that used in Ethernet. See the discussion of frame format in the section, "Ethernet Network Architecture."

Format of the Data

The data format used in StarLAN networks is identical to that used in Ethernet. See the discussion of data format in the section, "Ethernet Network Architecture."

Assignment of Network Addresses

The assignment of network addresses in StarLAN is similar to that for Ethernet. See the discussion on the assignment of network addresses in "Ethernet Network Architecture."

ARCNET® Network Architecture

ARCNET is a LAN suitable for moderate-speed interconnection of computers and computer-controlled devices over moderate to large distances. The architecture of ARCNET was originally defined by products available from Datapoint in 1977. Thus, it was arguably the first commercially available LAN. Implementations have been simplified and standardized since 1982 by the availability of an ARCNET LAN controller and associated devices from Standard Microsystems Corporation. There is no de jure standardization for ARCNET.

Physical Interconnection and Speed

ARCNETs use a token-bus topology and are connected so that every station hears what any station transmits. The delay between transmission and reception depends only on the propagation delays through the wires and attaching devices. In this way ARCNET differs fundamentally from a network like the IEEE 802.5 token ring where stations are wired in a logical ring so that each station only hears what its upstream neighbor transmits.

The ARCNET transmission speed is 2.5 megabits per second, sent in baseband (non-modulated, non-RF) form. Each data byte is preceded by a three-bit starting sequence, so that it takes 11 bits for each byte. The useful data rate is, therefore, 8/11 of 2.5 Mbps, which is 1.8 Mbps or 227,000 bytes per second.

Although the ARCNET network is logically a bus because every station hears what every other station transmits, it is physically wired as interconnected stars. Each station is wired to a repeater unit called a *hub.* Each hub can accept connections from 6 to 16 stations or other hubs. The topology is unconstrained except that there can be no loops, and no two stations may be separated by more than ten hubs.

The cable used to connect stations to hubs and between hubs is RG-62, 93-ohm coaxial cable. Since each cable connects to a hub on one end and a station or hub on the other end, the terminators are typically built into the adapters. No separate terminating devices are needed.

The maximum distance between a station and the hub, or between hubs, is 610 meters. Since stations can be up to ten hubs apart, the largest "diameter" of an ARCNET network is, thus, 6,710 meters.

There are two optional techniques for reducing the number of active hub ports needed to connect devices to an ARCNET network:

- One is by the use of a *passive* mini-hub, an inexpensive resistive coupler with three or four ports. Passive hubs function logically the same as the more expensive *active* hubs, but may not be connected directly to other passive hubs and may attach to cables no longer than about 61 meters. Passive hubs are commonly used when a single cable from an active hub needs to connect to two or three computers in a single room.

- More recently, a *multi-drop* or *high impedance* version of the network interface transceiver has been developed. With this technique a series of computers can be connected to a single cable from a hub in daisy-chain fashion, and a single

terminator is present at the end of the cable. The cable length is typically limited to 305 meters and the number of devices to ten.

Logical Interconnection and Access Control

Although ARCNET networks are wired in an almost arbitrary fashion, stations (not hubs) in the network are logically considered to be connected in a ring. Each of the stations on a single network is assigned an 8-bit address, typically by setting mechanical switches on the adapter card, but sometimes by the software driver. The logical ring consists of stations considered in the numerical order of their station addresses, which has nothing to do with their physical locations or the manner in which they are cabled to the network.

When there is no data traffic on the network, there is a continual transmission of the *token message* (see the first message in Figure 10), also called the *invitation to transmit message*. The station that has just received the token may, if it wishes, transmit a single data frame to any other station on the network. After that data frame is complete (or immediately after it received the token, if it has no data to transmit), the station must then send the token to the station on the network that has the next higher station address. It takes about 30 microseconds (plus cable propagation delays) for a station that has nothing to transmit to send the token to the next station.

Remember that all stations can hear what any station transmits, but the token-passing scheme insures that only one station is transmitting at a time. Each station listens for two kinds of transmissions:

- A *data message* (see the fifth message in Figure 10) being sent to it from any station on the network that just got the token and wants to send it data.
- The token sent to it from the station that has the next lower station address.

The logic within the hubs depends on the fact that there is only one transmitter at a time in order to be able to allow two-way communication using a single cable. In the idle state, a hub listens for incoming signals on all its ports. As soon as it detects the start of a packet (token or data) on any port, it switches all the other ports from input to output and retransmits the incoming packet out on all the other ports. When transmission is complete, it again enters the idle state to wait for another packet from any port.

Some newer ARCNET adapters violate the original transmission rules in order to increase performance. The most common variation, a *nodal priority scheme*, allows a station that has first transmitted to send again (perhaps by sending a token to itself) without waiting for the token to complete another round trip through the network.

Establishing and Maintaining the Token Sequence

During normal operation, the only information each station knows about the rest of the network is the address of next higher-addressed station, to which it must send the token. But how is the token passing started (or restarted after an error), and how does each station discover its neighbor's address after a start or a restart?

The elegantly simple process by which the token rotation is established is called a *reconfiguration*. Whenever there is no network activity for more than 78 microseconds, all stations assume that the token doesn't exist, and each starts an internal countdown based on the value of its own station address. The first station to count down to zero is the one that generates the initial token. It is first sent to an address one higher than the station's own address in the sequence 0, 1,2, ..., 254, 255, 0, 1, 2... If there is no activity after 74 microseconds, the conclusion is that there is no station at that address, and the next higher address is tried. This continues until a station is found, at which point the first station remembers that address as his next higher neighbor, and the neighboring station starts the search for his next higher neighbor. These searches continue until the station who originated the token receives it from his lower-addressed neighbor. At that point each station knows his higher neighbor, and normal token passing and data traffic can continue.

The entire reconfiguration process is quite fast. It can take as little as 24 milliseconds and never more than 61 milliseconds. It can occur anytime that the token has been destroyed by an error, but it normally occurs only when a new station joins the network. In that case, there is a token circulating, but it is never sent to the new station. When any station detects this situation (more than 840 milliseconds of network traffic during which it never received the token), it forces a reconfiguration by sending, without having received the token, a burst of data longer than any possible packet plus the subsequent token. That guarantees that the token will be destroyed, and then the reconfiguration algorithm will start and assign the token rotation sequence to include the new station.

Note that a full reconfiguration need not occur when a station leaves the network. In that case, the station that passes the token to the now-absent station will notice that neither token nor data is transmitted by that station within 74 microseconds after passing the token to it, so it will start sending the token to higher addresses until it finds the absent station's next neighbor and makes it his own.

When the hardware is working and is configured correctly, the algorithms for the maintenance of ARCNET token passing work extremely well, even when changes are being made to a live network. When two stations have the same address, however, or if any of several of hardware faults occur (such as a station whose receiver is defective but whose transmitter and controller are working), the network will reconfigure excessively. An effective technique for isolating the problem is to break the connection between hubs; the subnetwork that contains the fault will continue to reconfigure forever, but the other will become a correctly-operating network.

Transmitting Data Frames

One advantage that ARCNET has over most other LAN designs is that it avoids sending data on the network that cannot be accepted by the destination station. In the discussion above, the operation of sending a data frame actually consists of several steps. First, the station that just received the token transmits a *free buffer inquiry message* (see the second message in Figure 10) to the station to which it wishes to transmit data. There are several possible outcomes:

- If the receiver is present on the network and has a free buffer for the message, it sends back an *acknowledgement message* (see the third message in Figure 10), and then the data is sent.
- If the receiver exists on the network but has no buffer, it sends a *negative acknowledgement message* (see the fourth message in Figure 10), and the transmitter postpones sending the data until the next time it gets the token.
- If the station doesn't exist at all, the transmitter will hear no activity for 74 microseconds, then abandon the transmission, and pass on the token.

A second advantage of ARCNET not shared by most other networks is that there is an immediate confirmation at the data-link level that that the message was correctly received: the destination sends another *acknowledgement message* following the data. If the data was garbled, it sends nothing. After 74 microseconds with no response from the intended receiver, the transmitter knows that the message was not received correctly.

Format of a Frame

There are five messages that may appear on the ARCNET:

1. Invitation to transmit message or "token"

Alert: *6 bits*	Type code, hex 04: *1 byte*	Destination: *1 byte*	Destination: *1 byte*

2. Free buffer enquiry message

Alert: *6 bits*	Type code, hex 05: *1 byte*	Destination: *1 byte*	Destination: *1 byte*

3. Acknowledgement message

Alert: *6 bits*	Type code, hex 06: *1 byte*

4. Negative-acknowledgement message

Alert: *6 bits*	Type code, hex 15: *1 byte*

5. Data message

Alert: *6 bits*	Type code, hex 01: *1 byte*	Source: *1 byte*	Destination: *1 byte*	Destination: *1 byte*	Count: *1-2 bytes*	Data: *1-508 bytes*	CRC: *2 bytes*

Figure 10. Five ARCNET messages.

Each field used in the five ARCNET messages is described below:

- **Alert.** A 6-bit announcement that a message is to follow.
- **Type code.** Indicates which one of the five types of messages it is. Each message type contains only the necessary information. Note, for example, that the token does not contain the address of the station it is *from* but only the station it is *to*.
- **Source.** The address of the station originating a data message.

- **Destination**. The address of the station slated to receive the data message. The repetition of the destination address is an artifact of the original implementation and is not strictly necessary.

- **Count**. The count field of the data message is a complicated encoding of the number of bytes in the packet that is another holdover from the details of original ARCNET implementation. For data lengths (N) from 1 to 253 bytes (i.e., "short packets"), the count field is a single byte whose value is 256-N. For data lengths from 257 to 508 bytes (i.e., "long packets"), the count field is two bytes. The first is zero, and the second is 512-N. Frames with data lengths from 254 to 256 cannot be sent at all.

- **Data**. Up to 508 bytes of data starting with the **System ID** (see Figure 11).

- **CRC**. The CRC at the end of the data frame is a two-byte *cyclic redundancy check* that the receiver uses to verify that the data has been received correctly.

Format of the Data

The Sniffer analyzer records only data messages and, thus, avoids having the buffer cluttered with the other messages not concerned with higher-level protocols.

The Sniffer analyzer records data frames in a "normalized" fashion that simplifies presentation of the data.

Source:	**Destination:**	**Length:**	**System ID:**	**Protocol Data:**
1 byte	*1 byte*	*2 bytes*	*1 byte*	*0-507 bytes*

Figure 11. An ARCNET data frame as recorded and as "normalized" by the Sniffer analyzer.

The length field indicates the number of data bytes that follow. Thus, fields that are at the same offset in the data in both long and short frames will appear in the same position in the displayed frame.

- **System ID**. A convention established by Datapoint uses the first byte of the data field to indicate what first-level protocol is being used. System IDs from hex 00 to hex 7F are reserved for Datapoint. IDs from hex 80 to hex FF are assigned to other organizations by Datapoint upon request.

- **Protocol data**. Up to 508 bytes of data starting with the **System ID**.

IBM PC Network™ (Broadband) Architecture

The IBM PC Network (Broadband) is a LAN suitable for interconnecting computers and computer-controlled devices at moderate speeds over moderate distances. The architecture of PC Network was defined jointly by IBM and Sytek in 1984. It is not described by any of the existing international standards and is not likely to be.

The original name for this network was simply PC Network, and it applied both to the hardware and to software protocols up to the application level. The full name is now IBM PC Network (Broadband), and it applies to the network hardware and data link control protocol but not any of the higher protocols. Note that there is also now a baseband version of PC Network, but we will use "PC Network" to mean only the "IBM PC Network (Broadband)."

Speed and Transmission Technique

Every station connected to a PC Network can hear what any station transmits. The delay between transmission and reception depends only on the propagation delays through the wires and the various attaching devices. In this way, PC Network is similar to networks like Ethernet and StarLAN (see the sections, "Ethernet Network Architecture" and "StarLAN Network Architecture"). On the other hand, it differs fundamentally from networks like the IEEE 802.5 token ring, where stations are wired in a logical ring so that each station only hears what its upstream neighbor transmits (see the section,"Token Ring Network Architecture"). The transmission speed on PC Network is 2 Mbps, or 250,000 bytes per second.

The bits to be sent are not simply placed on the network cable but are used to modulate a high-frequency carrier signal by using the *frequency-shift-keying* (FSK) technique. This use of *radio frequency* (RF) transmission is what characterizes PC Network as a broadband network when compared to baseband networks such as Ethernet.

As with all two-directional broadband systems, nodes transmit at a relatively low frequency but listen at a much higher frequency. There is a special device on the network called a *translator* or *headend*. The translator converts a low-frequency transmission by any station into an equivalent high-frequency signal that all stations hear. The frequency difference between the two signals is called the *offset*. Note that unlike dual-cable broadband systems, both the original low-frequency signal and the translated high-frequency signal are present on the same cable at the same time. The cable can also carry completely unrelated information on other channels. Two examples are closed-circuit TV transmissions used by plant security in some installations and standard cable TV signals.

The total bandwidth used by a single transmission is 12 MHz (one 6 MHz transmit channel and one 6 MHz receive channel). This applies to the entire PC Network since all stations typically use the same frequencies. It is the same frequency-division multiplexing scheme used by CATV systems. The entire PC Network uses the equivalent of only two TV channels on the cable. Whether the network is compatible with a particular CATV system depends on the choice of carrier frequencies.

Broadband systems differ in the amount of cable bandwidth devoted to *forward* traffic (from the headend to nodes) versus *return* traffic (from nodes to the headend). Systems designed primarily for television are typically *low-split* so that most traffic is in the forward direction. Computer-oriented systems are *mid-split* or *high-split* and allow approximately equal traffic in both directions.

There are four different versions of the PC Network adapter that differ only in their frequency assignments. The numbers in the first two columns of Figure 12 are the center frequencies of the 6 MHz channels in megahertz, and the symbols in parentheses are the channel names:

Transmit	Receive	Offset
50.75 (T14)	219.00 (J)	168.25
56.75 (2')	249.00 (O)	192.25
62.75 (3')	255.00 (P)	192.25
50.75 (T14)	243.00 (N)	192.25

Figure 12. Frequency assignments for different versions of the PC Network adapter.

The assignments on the first line are those for the original 1984 PC Network, which used a non-standard offset. The other assignments were added later and use an offset that is also the standard for MAP (IEEE 802.4) networks. All the adapters in a single network must use the same carrier frequencies, although multiple independent networks may operate at different frequencies simultaneously on the same cable system.

Physical Interconnection

Stations are connected to an attachment point of the network using standard CATV hardware: RG-59, 75-ohm coaxial cable and F-connectors. For higher reliability, the screw-on type connectors should be used, although a push-on connector will work as long as it makes a tight connection.

Station cables typically attach to a device called a *signal splitter, directional coupler,* or *attenuator.* The usual topology is a tree with the headend at the root and splitters branching off to other splitters or nodes. It is also possible to use a more bus-oriented topology with the headend at a less central location.

A broadband cable system must be carefully designed so that both receive and transmit signals have the proper amplitude at every node and at the headend. For small networks there are *kits* available for various combinations of cable distances and number of nodes. Kits available for the PC Network limit the number of stations to 72 and the maximum radius of the network to 1000 feet.

Larger or unusual networks must be custom-designed using carefully chosen splitters and attenuators and may require special RF test equipment for setup and maintenance. Among the problems to be solved is that different frequencies are attenuated by different amounts. To cope with this, the designer must use a frequency-dependent attenuator called a *tilt compensator* or *equalizer.*

Access Control

PC Network LANs, like Ethernet and StarLAN, use the CSMA/CD algorithm to control transmission. See the discussion of access control in the section, "Ethernet Network Architecture."

Format of a Frame

The frame format used in PC Network is very similar to that used in Ethernet. See the discussion of frame format in the section, "Ethernet Network Architecture."

Format of the Data

The data format used in PC Network is very similar to that used in Ethernet. See the discussion of data format in the section, "Ethernet Network Architecture."

As of this writing, the Sniffer analyzer does not decode the original NetBIOS format frames.

Assignment of Network Addresses

The assignment of network addresses in PC Network is similar in some ways to that in Ethernet. See the discussion on the assignment of network addresses in the section, "Ethernet Network Architecture."

LocalTalk® Network Architecture

LocalTalk is a type of LAN suitable for low-speed interconnection of computers and computer-controlled devices over short distances. The architecture of LocalTalk is defined de facto by Apple Computer, Inc.

Physical Interconnection and Speed

Stations connected to a LocalTalk network are all connected to the same wire, so that every station hears what any station transmits. The delay between transmission and reception depends only on the propagation delays through the wires and attaching devices. In this way LocalTalk is similar to networks like Ethernet, StarLAN and IBM PC Network (see the sections, "Ethernet Network Architecture," "StarLAN Network Architecture," and "IBM PC Network Architecture"). On the other hand, it differs fundamentally from networks like the IEEE 802.5 token ring, where stations are wired in a logical ring so that each station only hears what its upstream neighbor transmits (see "Token Ring Network Architecture").

The LocalTalk transmission speed is 230.4 Kilobits per second (Kbps), or 28,800 bytes per second. LocalTalk's signalling standard is EIA modified RS-422 (balanced voltage), and signal encoding uses a technique known as FM-0 (also called *biphase space*). Because of the access control mechanism described below, the effective network throughput is significantly less than the transmission speed.

LocalTalk is a *bus* network system. A *connector box* is attached to all devices on the network with either a DIN-8 mini-circular plug or DB-9 plug, depending upon the device. The connector boxes are then connected to each other with LocalTalk cable equipped with DIN-8 plugs. LocalTalk cabling consists of shielded twisted pair 22 AWG stranded wire.

The maximum network cable length is 300 meters. The maximum number of connector boxes in any single network is 32.

Access Control

LocalTalk uses an access discipline to control transmission called *carrier sense multiple access with collision avoidance* (CSMA/CA). A similar architecture is *carrier sense multiple access with collision detection* (CSMA/CD), implemented in Ethernet and its variants, StarLAN and PC Network (see "Ethernet Network Architecture").

Like the CSMA/CD technique, CSMA/CA requires each node wishing to transmit data to check the transmission medium before sending (e.g., carrier sense) and permits more than one node to access the link (e.g., multiple access), but not simultaneously. Because of multiple access, both techniques run the risk of two or more nodes attempting to transmit at the same time and having their respective frames collide. The difference between how each handles this situation is significant.

With CSMA/CA, hardware does not actually detect collisions. Rather, collisions are inferred through the use of a "handshake" mechanism. The handshake

involves a transmission dialog using LocalTalk Link Access Protocol (LLAP) control packets prior to sending any data. The transmission dialog begins when the transmitting node's physical layer senses that the line is idle. After sensing an idle line, the transmitting node has a wait period of 400 microseconds plus a randomly generated period. It then sends an LLAP *request-to-send* (RTS) control packet to the intended receiver. The transmitter waits up to 200 microsecond for an LLAP *clear-to-send* (CTS) control packet from the receiver.

If the transmitter receives the CTS packet, it considers the handshake successful. A successful handshake means that a collision did not occur. All other nodes defer 200 microseconds for the transmitter to send its data packet.

If the transmitter does not receive the CTS packet within the allotted 200 microseconds, the transmitter backs off and retries after a random wait period. The transmitter assumes that a collision took place and retries for up to 32 times before reporting failure to its client. If in fact the RTS packet did collide, the receiving node uses the frame check sequence to discover the corrupted packet and to discard it.

The transmission dialog starts similarly for broadcast packets. But after a transmitter sends an RTS packet with the destination address set to all nodes, it does not expect to receive a CTS packet. Rather, it continues to sense the line. If the line remains idle for 200 microseconds, the transmitter sends its data frame.

If the broadcast RTS packet forces a collision, the other transmitting node will back off. The broadcasting node delays until the link is idle again and until the wait period is over. It tries again for up to 32 times before reporting failure to its client, although collisions with other broadcasting nodes may not be noticed.

Format of a Frame

A LocalTalk network uses the LLAP at the data-link layer to allow network devices to share the communication medium. The LLAP frame **preamble** identifies the start of the frame with *flag* bytes. A **flag** byte also identifies the end of the frame. A flag is a frame delimiter that is identifiable with a special bit sequence, `01111110`. Sometimes a flag-like sequence will be inserted into the user data stream by the application process and creates the possibility for an error. To protect against such errors, LLAP uses a technique known as *bit-stuffing* (see the discussion of bit-stuffing in the section, ("WAN/Synchronous Network Architecture").

Preamble: *2 or more flag bytes*	**Destination Node ID:** *1 byte*	**Source Node ID:** *1 byte*	**LLAP Type:** *1 byte*	**Protocol Data:** *2 to 600 bytes*	**FCS:** *2 bytes*	**Flag:** *1 byte*	**Abort Sequence:** *12 to 18 1s*

Figure 13. LLAP frame format.

LLAP uses a 1-byte node identifier number to identify each node on a link. For transmitting and receiving stations, these serve as **source node** and **destination node** data-link addresses.

The **LLAP type** field specifies either a control packet or a data packet. There are four types of control packet as listed in Figure 14. Control packets do not have a data field. Two are concerned with the dynamic assignment of addresses in

LocalTalk. Unlike other network data links, LocalTalk nodes do not have fixed unique addresses. Nodes newly activated on a network choose an address for themselves. They use an **enquiry** packet to verify if the address they chose is unique. If it is already in use, the node with that address responds with an **acknowledgement** packet, and the new node tries again with another enquiry.

lapENQ	hex 81	*Enquiry.* Used for dynamic assignment of node address.
lapACK	hex 82	*Acknowledgment.* Responds to an *Enquiry* packet.
lapRTS	hex 84	*Request-to-send.* Notifies destination node that a data packet is ready.
lapCTS	hex 85	*Clear-to-send.* Responds to RTS indicating readiness to accept packet.

Figure 14. Indications used in LLAP control frames.

The other two control packets are part of LLAP's RTS-CTS handshake mechanism. As noted above in "Access Control," RTS-CTS handshake is successful if the node transmitting the **request-to-send** receives a **clear-to-send** packet from the receiving node in the appropriate time. The handshake is not successful when the CTS packet is not received as expected.

Format of the Data

If the packet type field indicates a data packet, then the first two bytes of the data field must contain **data length** information in bytes.

Figure 15. LLAP data frame format.

The low-order 10 bits of the data length field contain the size in bytes (most-significant bits first) of the data field, including the two data length bytes. Higher-level protocols reserve the use of the high-order 6 bits of the data length field.

WAN/Synchronous Architecture

X.25 protocols, running over a synchronous serial link, provide the most widely implemented public data network access method. X.25 defines the general purpose interface between *data terminal equipment* (DTE) and *data circuit-terminating* (or *communications*) *equipment* (DCE). A DTE is an end-user machine. The entry point to the *wide area network* (WAN) is a DCE.

Several standards have been implemented at the lower layers of the synchronous architecture and allow users a wide range of options to interface to a packet-switched network. Physical level standards include the Electrical Industries Association's familiar RS-232C. The International Consultative Committee for Telephony and Telegraphy (CCITT) defined two well-known sets of standards, the V-Series (V.35, for example) and the X-Series (X.21, for example). Several of these other standards are related to RS-232C: CCITT V.24 and V.28; CCITT X.21bis ; and ISO 2110.

At the link level, the International Organization for Standardization (ISO) published the widely used standard, High-level Data Link Control (HDLC) protocol. There are several important protocol subsets of the HDLC superset. Link Access Procedure, Balanced (LAPB) supports the widely accepted X.25 packet network protocol. Synchronous Data Link Control (SDLC) is IBM's version of HDLC and is used for its Systems Network Architecture (SNA) protocol. Finally, LLC is the standard released by the IEEE 802 standards committee for LANs (See the discussion of "LLC Frames" in the section on Ethernet network architecture).

Interconnection and Speed

Typically, RS-232C specifies a 25-pin connector, so that up to 25 wires can be used to connect two devices. The electrical characteristics of RS-232C place a limit of about 50 feet on the distance between, for example, a DTE and its modem. These same characteristics limit the data transmission rate across the interface to a maximum of about 19.2 kilobits per second.

V.35 allows data transmission at higher speeds (typically, 56 kilobits per second). It is implemented using both the frequency modulation and amplitude modulation techniques. The Sniffer connection to V.35 is baseband.

Format of a Frame

Frames captured from a synchronous line and interpreted by the WAN/Synchronous Sniffer analyzer generally conform to the HDLC standard defined by ISO. In particular, two subsets of HDLC are relevant to WAN/Synchronous Sniffer users: SDLC and LAPB. The Sniffer analyzer will also recognize a proprietary version of HDLC framing implemented by cisco Systems.

LAPB is the link layer protocol for the network layer protocol, X.25. Sniffer menus and screens refer to LAPB as HDLC. SDLC is IBM's version of the HDLC superset. It is the link layer protocol for IBM's network layer services of its SNA protocol.

One major difference between LAPB and SDLC has to do with stations' responsibilities and the rules stations follow when transferring information. LAPB stations, like LLC stations, allow any station to initiate transmissions without prior permission from any other station (referred to as *Asynchronous Balanced Mode* or ABM). SDLC, on the other hand, requires a subservient *secondary* station to receive explicit permission from a dominant *primary* station before transmitting (referred to as *Normal Response Mode* or NRM). Unlike the *balanced* configuration of LAPB stations where stations are equally responsible for the link, the SDLC station configuration is *unbalanced*—the prime responsibility for the link depends upon the primary station.

Both LAPB and SDLC conform to the general HDLC frame format. The format of each frame is as follows:

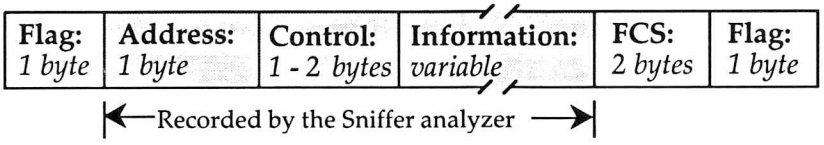

Figure 16. The general HDLC (including LAPB and SDLC) frame format.

The **flag** is a special bit sequence, `01111110`, that indicates either the beginning or the end of a frame. Sometimes a flag-like sequence will be inserted into the user data stream by the application process and creates the possibility for an error.

To insure that the sequence is not interpreted as an opening or closing flag of a frame, and, thus, to prevent errors, HDLC uses a technique called *bit-stuffing*. The transmitting machine checks between the opening and closing flags, and it inserts a `0` bit when it encounters five continuous `1` bits. After the frame has been "stuffed" and flags placed, the transmitting machine sends the frame to the receiver. The receiver monitors the bit stream. When the receiver encounters a 0 bit followed by five continuous `1` bits, it checks the seventh bit. If the seventh bit is a `0`, the receiver "unstuffs" that bit and looks at the eighth bit. If the eighth bit is a `1`, it inspects the ninth bit. The receiver knows the bit sequence is a flag if the ninth bit is a `0`. However, if the ninth bit is a `1`, then it knows it is either an abort or an idle signal and takes appropriate action.

The **address** field identifies the primary or the secondary station transmitting a particular frame. Each station has a unique address. In unbalanced configurations, address fields contain addresses of secondary stations in both commands and responses. In balanced configurations, command frames contain the destination address, and response frames contain the transmitting station address.

The **control** field defines the exact function of LAPB and SDLC frames using the general HDLC frame format. It performs various functions, depending upon whether a frame is an *Information*, *Supervisory*, or *Unnumbered* frame (see below).

There are two important differences between LAPB and SDLC frames with regard to the control field:

- One is that each type of configuration has its own mode-setting command to establish link-level contact as either balanced or unbalanced.

- The other difference is that in the balanced configuration of LAPB, all stations can send commands and responses. In the unbalanced configuration of SDLC, a frame sent by a primary station is a command, and a frame sent sent by a secondary station is a response.

There are three HDLC frame types:

Information **I** format frame are used to transmit end-user data between two devices and to acknowledge receipt of data from a transmitting station. It also performs limited functions, like the *Poll* command. The control field of this type of frame contains both a *send sequence number* N(S) for itself and a *receive sequence number* N(R) of the next frame expected from the other station.

Supervisory **S** format frames are used to control the flow of data. The control field of this frame type contains an N(R) number but not an N(S) number. In addition, the control field contains the the indications listed in Figure 17 as either a command or a response.

RR	*Receive Ready*. Transmission of Information frames can proceed.
RNR	*Receive Not Ready*. Transmission is temporarily blocked.
REJ	*Reject*. Retransmission starting with N(R) number is requested.

Figure 17. Commands and responses used in HDLC (SDLC and LAPB) supervisory frames.

Unnumbered **U** format frames are also used for control purposes. The control field of this frame type contains neither N(S) nor N(R) numbers but may contain control information or data. The commands and responses appearing in the unnumbered frame control field are listed in Figure 18.

SNRM	*Set Normal Response Mode.* Configure a secondary station in a mode that precludes it from sending unsolicited frames. The primary station controls all message flow. Used in SDLC.
SNRME	*Set Normal Response Mode (Extended).* Same as SNRM except that it selects modulo 128 sequence numbering rather than modulo 8 sequence numbering.
SABM	*Set Asynchronous Balanced Mode.* Configure stations as peers with each other. No polls are required to transmit. Used in LAPB.
SABME	*Set Asynchronous Balanced Mode (Extended).* Same as SABM except that it selects modulo 128 sequence numbering rather than modulo 8 sequence numbering.
DISC	*Disconnect.* Command used to place a secondary station in the disconnected mode (not operational).
DM	*Disconnected Mode.* Used by secondary station to indicate that it is in the disconnected mode.
UA	*Unnumbered Acknowledgement.* Response to set mode commands and to DISC.
FRMR	*Frame Reject.* The format of a received frame was invalid. Information field contains the reason.
XID	*Exchange Identification.* Used between two stations to exchange identification and the characteristics of the two stations.
UI	*Unnumbered Information.* Used for transmission of user data in an unsequenced frame.

Figure 18. Commands and responses in HDLC (SDLC and LAPB) unnumbered frames.

The only HDLC frames that are allowed to contain data after the header are I, UI, TEST, XID, and FRMR types.

Phases of Link Control

The following five phases represent the fundamental sets of activities on a synchronous line using HDLC (SDLC and LAPB):

Connect phase

Establishes a connection over a switched facility. The process includes off-hook signalling, switching, and exchange of identification.

Link establishment phase

This is the first phase in the span of link control protocols. Typical processes include initializing data transfer over an already established physical link and polling.

For example, when a DCE is able to proceed with a connection, it sends DISC frames with the *Poll* bit set on. When a DTE wants to reconnect to the DCE, it waits until it receives a DISC frame and then responds with a UA frame with the *Final* bit

set on. The DCE is responsible for setting up the link when it receives the UA frame within the required amount of time. The specific link setup command depends upon the HDLC subset in use: if it is LAPB, the commands are SABM and SABME; if it is SDLC, the commands are SNRM and SNRME. After the DCE receives the link setup command in the required amount of time, it responds with a UA frame, and the link is now in an "up" state.

Information transfer phase

The information transfer phase includes processes associated with the transfer of data. It begins following link establishment and terminates with the end of the message or data transfer. It includes the actual data transfer between connected stations as well as the acknowledgement process.

For example, the DTE sends an I frame. The DCE may acknowledge with either an RR frame or another I frame, and it may not acknowledge right away. When the DCE does acknowledge, it includes an N(R) number that is inclusive of all traffic transmitted and accepted since the last acknowledgment.

Termination phase

The termination phase relinquishes control of the link following transmission of data. In an unbalanced configuration, the secondary station returns control to the primary station.

Clear phase

The clear phase releases the facility. For example, the DTE sends a DISC frame to clear down a link without the Poll bit set on. The DCE responds by transmitting a UA frame without the Final bit set.

2. Major Protocol Suites

2

Network General

Chapter 2. Major Protocol Suites

Chapter 2. Major Protocol Suites

Chapter Overview

The Sniffer analyzer decodes twelve major protocol suites. Following a brief introductory section on the OSI layered protocol model, Chapter 2 provides some general information on each of the suites. The information includes a short discussion of the suite itself, a diagram that maps each protocol to the OSI model, and a summary of the services provided by each protocol.

Introduction

The Sniffer network analyzer does not simply monitor, capture, or record network traffic. It also interprets what it records. Protocol interpretation is what makes the Sniffer analyzer a valuable tool. Interpretation turns an inscrutable stream of bits and bytes into clearly labeled commands, responses, and readable text.

Interpretation routines are an integral part of the Sniffer software, built-in at the factory during a custom compilation that equips each machine for the network and protocols requested by the customer. Network General Corporation calls the interpretation routines *protocol interpreters* (PI). The interpretation they do covers the full range of the Open Systems Interconnection (OSI) seven-layer model.

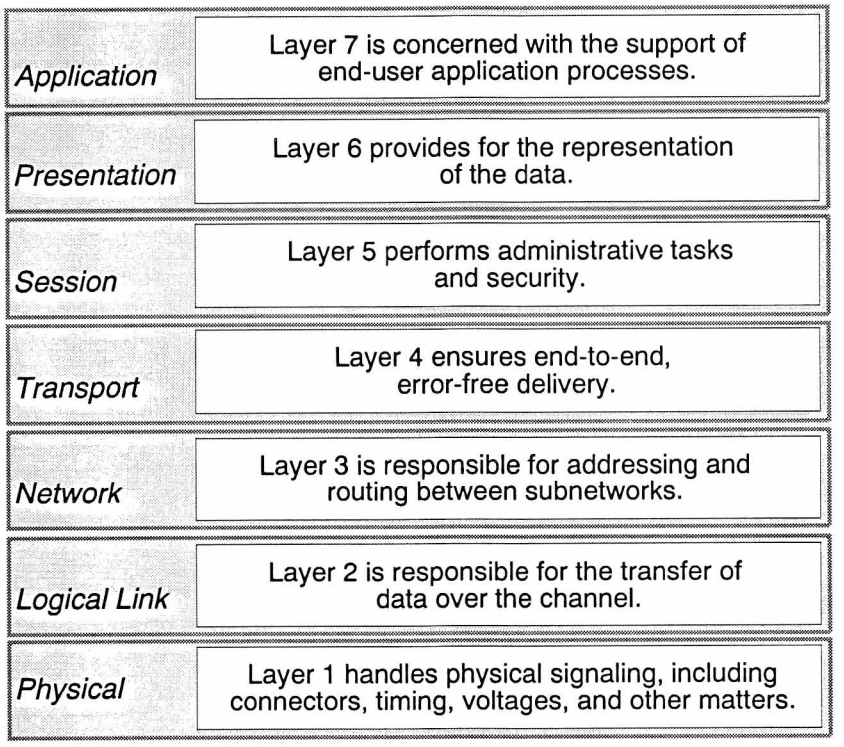

Figure 19. Layers of the OSI Network Model.

At the lowest *physical* layer, the analyzer's hardware is responsible for sending and receiving network signals. For the next layer, *logical link control*, interpreters are included to match the networks on which the analyzer will be used.

For protocols at any of the higher layers—*network, transport, session, presentation,* or *application*—interpreters are included for one or more of the many optional protocol interpreters that may be ordered with the basic unit. A network's upper-level protocols are largely independent of its physical layer. Each of the protocol suites covered in this chapter are found on a variety of Sniffer analyzers equipped for different types of network.

Figure 20 lists PIs included with the Sniffer analyzer regardless of which of the optional PI suites have been included, although they are also shown in the appropriate suite diagrams.

Network	PI	Function
All networks	DLC	Data Link Control. Physical level protocol corresponding to the network type.
	LLC	Logical Link Control. 802.2 protocol.
	RI	Routing Information. Used for source routing.
	BPDU	Bridge Protocol Data Unit. 802.1 spanning-tree bridge protocol.
	SNAP	Sub-Network Access Protocol. Used to embed non-LLC protocols (typically Ethertypes) within LLC 802.2.
	LOOP	Loopback Protocol. Ethernet-style loopback test protocol.
Token ring	MAC	Medium Access Control Protocol. Used in 802.5.

Figure 20. Sniffer PIs included regardless of PI suites installed

IBM® Protocol Interpreter Suite

The Sniffer analyzer interprets frames in four families of higher-level protocols widely used on IBM networks. While IBM uses these protocols on LANs connected by token ring, they may also be found on networks connected by other media. The IBM PI suite may be installed in a Sniffer unit equipped for connection to a LAN such as token ring, PC Network (Broadband), Ethernet, or StarLAN or to a wide area network's WAN/synchronous link by way of an RS-232 or V.35 interface.

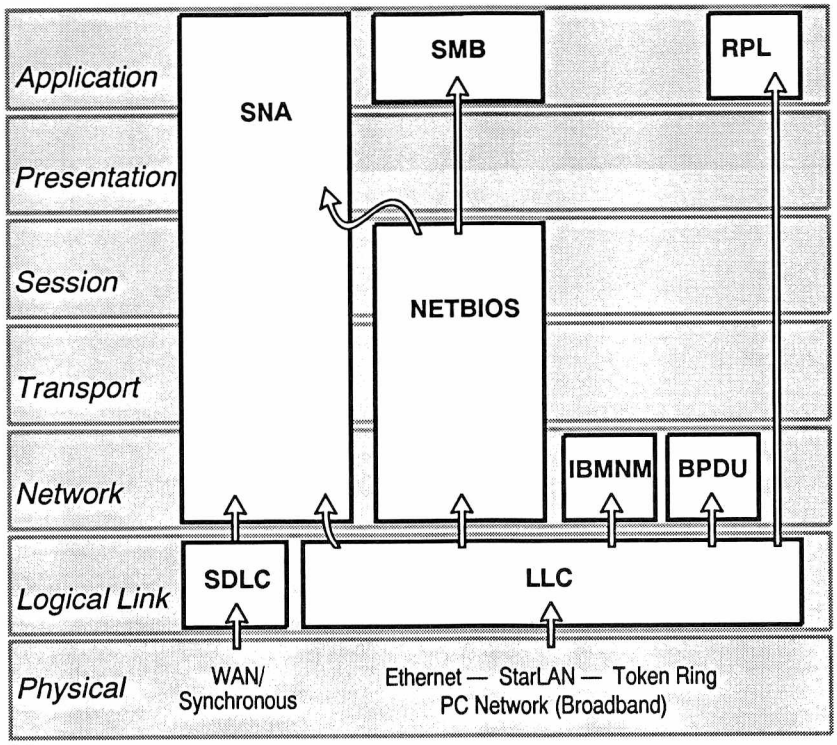

Figure 21. The IBM PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

SNA *Systems Network Architecture.* IBM's name for its very extensive family of commands within a common protocol, widely used to connect a broad range of devices, from terminal controllers to micro- or minicomputers, to IBM mainframes. The interpreter decodes the SNA transmission header, the request/response header, and the request and response content by area, including transmission services, function management, management services, presentation services, and general data stream. Both LU2.0 and LU6.2 commands are decoded.

SMB *Server Message Block.* A family of application-level commands for LAN servers developed by Microsoft® for use with the IBM PC LAN Program but frequently used in other environments as well. Many of the functions are similar to those made by an application program to DOS or to OS/2® running on a single computer. The IBM PC LAN Program sends SMBs as data within NetBIOS frames, but in other contexts, they may be sent differently. The SMB protocol for machines running under OS/2 contains extensions not present in the version for DOS

machines. The PI decodes both the older DOS versions and the extended OS/2 versions.

RPL *Remote Program Load.* A protocol used by IBM on the IEEE 802.5 token ring network to download initial programs into networked stations.

NETBIOS *Network Basic I/O System.* A protocol implemented in the IBM PC LAN Program to support communication between symbolically named stations and the exchange of arbitrary data between symbolically named stations. (Some of the other NetBIOS implementations differ from the IBM version. The NETBIOS module of the IBM PI suite differs accordingly from the corresponding NetBIOS modules of Novell NetWare® and TCP/IP.)

IBMNM *IBM Network Management Protocols (LLC SAP F4).* Used for the LAN Reporting Mechanism, Ring Error Monitor, Configuration Report Server, Ring Parameter Server, and LAN Bridge Server.

BPDU *Bridge Protocol Data Units.* Used for the 802.1 spanning tree routing algorithm.

LLC *Logical Link Control (IEEE 802.2).* A protocol that provides connection control and/or multiplexing to subsequent embedded protocols.

SDLC *Synchronous Data Link Control.* IBM's version of the logical link layer protocol whose ISO designation is HDLC. The WAN/Synchronous Sniffer analyzer interprets the subset that provides link-level support for X.25 and SNA.

Novell® NetWare® Protocol Interpreter Suite

The Sniffer analyzer interprets the protocols used by Novell's NetWare family of products, which include an operating system for file servers as well as services in support of remote users on a variety of physical media. The interpreter suite may be installed on Sniffer systems for Ethernet, ARCNET, StarLAN, token ring, IBM PC Network (broadband), or WAN/synchronous.

Each NetWare server runs directly under a proprietary Novell operating system. Users at DOS- or OS/2-based workstations can redirect their operating system functions to the NetWare servers. What Novell calls Network File Services (NFS) make use of NetWare Core Protocol (NCP) to transmit commands or inquiries from workstations and to receive replies from servers. NCP in turn makes use of Novell's implementation of the XNS family of protocols developed by Xerox. These protocols are concerned with the transmission and delivery of a packet, but not its interpretation, which is left to the higher-level protocol, NCP.

At the network level, NetWare uses a datagram protocol called IPX that corresponds to Xerox's IDP (Internet Datagram Protocol). Each IPX packet identifies the network, node and socket of its destination and of its source. A socket may be a function within a node and, hence, affects where the embedded NCP message is interpreted.

NetWare also provides a connection-oriented virtual circuit protocol called SPX (Sequential Packet Exchange) that corresponds to SPP in the XNS protocols. (However, NCP provides connection services without the use of SPX packets.) In SPX, each packet is identified in the same way as an IPX packet, but with additional fields for the source and destination connection, a sequence number within that connection, an acknowledgment number, and an allocation of the number of unacknowledged SPX packets the connection may tolerate.

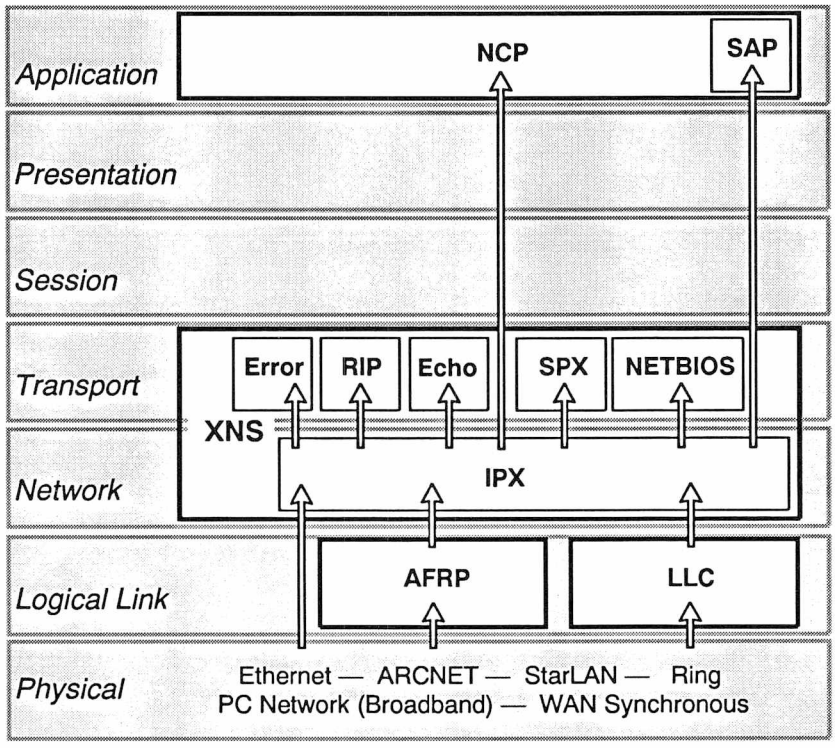

Figure 22. The Novell NetWare PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

NCP *NetWare Core Protocol.* Novell's application-level protocol for the exchange of commands and data between file servers and workstations; also described as NetWare File Service Protocol (NFSP).

SAP *Service Advertising Protocol.* Used by NetWare servers to broadcast the names and locations of servers and to send a specific response to any station that queries it.

NetBIOS *Network Basic I/O System.* NetWare supports emulation of the protocol implemented by the IBM PC LAN Program to support communication between symbolically named stations and the exchange of arbitrary data. In the NetWare context, NetBIOS is atop IPX.

XNS *Xerox Network Systems Protocols.* Within this family of protocols, the following are identified:

- ***SPX*** *Sequential Packet Exchange.* Novell's version of the Xerox transport-level protocol called SPP.
- ***IPX*** *Internet Packet Exchange.* This network-level protocol corresponds to Xerox IDP.
- ***RIP*** *Routing Information Protocol.* Novell's version of a protocol used to exchange routing information among gateways.
- ***Echo*** Request/response protocol used to verify the existence of a host.
- ***Error*** A protocol by which a station reports that it has received (and is discarding) a defective packet.

AFRP *ARCNET Fragmentation Protocol.* Breaks up and reassembles network-layer packets so that they are acceptable to the data-link protocol and the underlying physical medium.

LLC *Logical Link Control (IEEE 802.2).* A protocol that provides connection control and/or multiplexing to subsequent embedded protocols.

XNS™ Protocol Interpreter Suite

At the **network, transport** and **presentation** layers, the PI suite handles the protocols of Xerox Network System (XNS). After Xerox published the specifications of these protocols in 1981, several other vendors developed application-layer protocols that run on top of them. The XNS PI suite decodes SMB, a protocol used in Microsoft Networks (MS-NET™) and the IBM OS/2 LAN Manager™.

A network's upper-level protocols are largely independent of its physical layer. While Xerox developed XNS for operation with Ethernet systems, XNS protocols may also be found on networks connected by other media. The XNS PI suite may be installed in a Sniffer analyzer equipped for connection to Ethernet, StarLAN, token ring, IBM PC Network (Broadband), or WAN/synchronous.

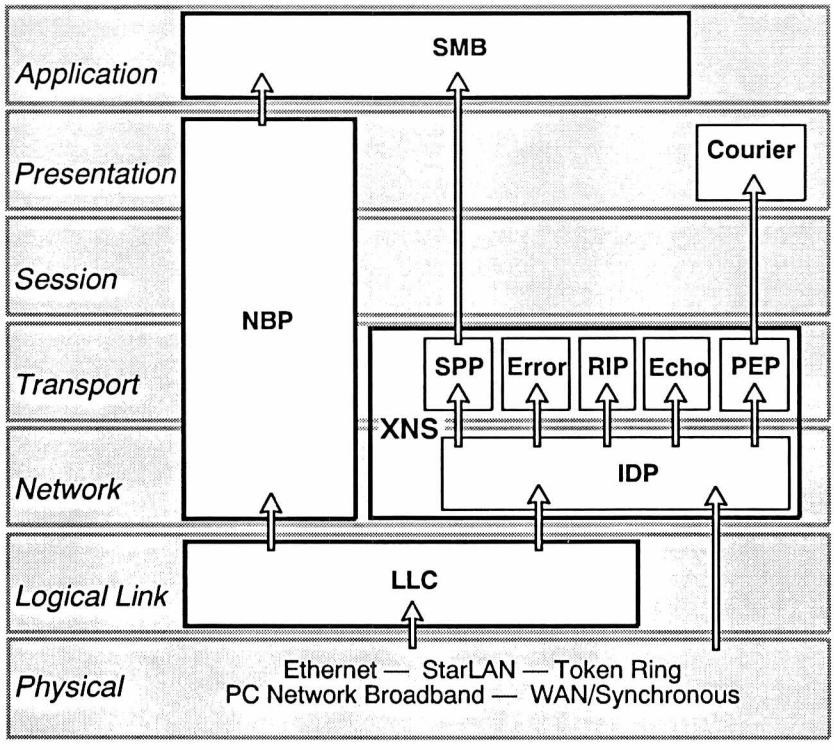

Figure 23. The XNS PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

SMB *Server Message Block.* A family of application-level commands for LAN servers developed by Microsoft and IBM for use with the IBM PC LAN Program but frequently used in other environments as well. Many of the functions are similar to those made by an application program to DOS or to OS/2 running on a single computer. Although the IBM PC LAN Program sends SMBs as data within NetBIOS frames, in other contexts they may be sent differently. The XNS PI suite decodes SMB frames transported by the Xerox IDP and SPP protocols. The SMB protocol for machines running under OS/2 contains extensions not present in the

version for DOS machines. The XNS PI suite decodes both the older DOS versions and the extended OS/2 versions.

NBP *NetBIOS Protocol.* Used in 3Com 3+ Open software.

XNS *Xerox Network Systems Protocol.* Within this family of protocols, the XNS PI suite interprets the following:

Courier A presentation-level protocol that delivers data to such application-level protocols as XNS Printing, XNS Filing, or XNS Clearinghouse (which the XNS PI suite identifies but does not interpret).

SPP *Sequenced Packet Protocol.* A virtual-circuit, connection-oriented protocol.

IDP *Internet Datagram Protocol.* Delivers to an internet address a single frame as an independent entity, without regard to other packets or to the addressee's response.

PEP *Packet Exchange Protocol.* Delivers a request and response pair; this protocol thus has a reliability greater than IDP alone, but less than achievable with SPP.

RIP *Routing Information Protocol.* Exchanges routing information among gateways and end systems.

Echo Request/response protocol used to verify the existence of a host.

Error Protocol by which a station reports that it has received (and is discarding) a defective packet.

TCP/IP Protocol Interpreter Suite

The Sniffer analyzer interprets the protocols of the TCP/IP family and other related protocols. TCP/IP was developed during the 1970's by research institutions under grants from the Advanced Research Projects Agency (ARPANET), US Defense Department. Since its adoption as a standard for ARPANET in 1978, TCP/IP has become widely used in many other networks linking commercial or educational institutions. Although the U.S. Congress has mandated the eventual adoption of ISO protocols, TCP/IP is likely to remain widely used for some time.

While TCP/IP usually runs on Ethernet, its protocols may also be found on other networks. The TCP/IP PI suite may be installed in a Sniffer system for Ethernet, ARCNET, StarLAN, token ring, PC Network (Broadband), or WAN/synchronous.

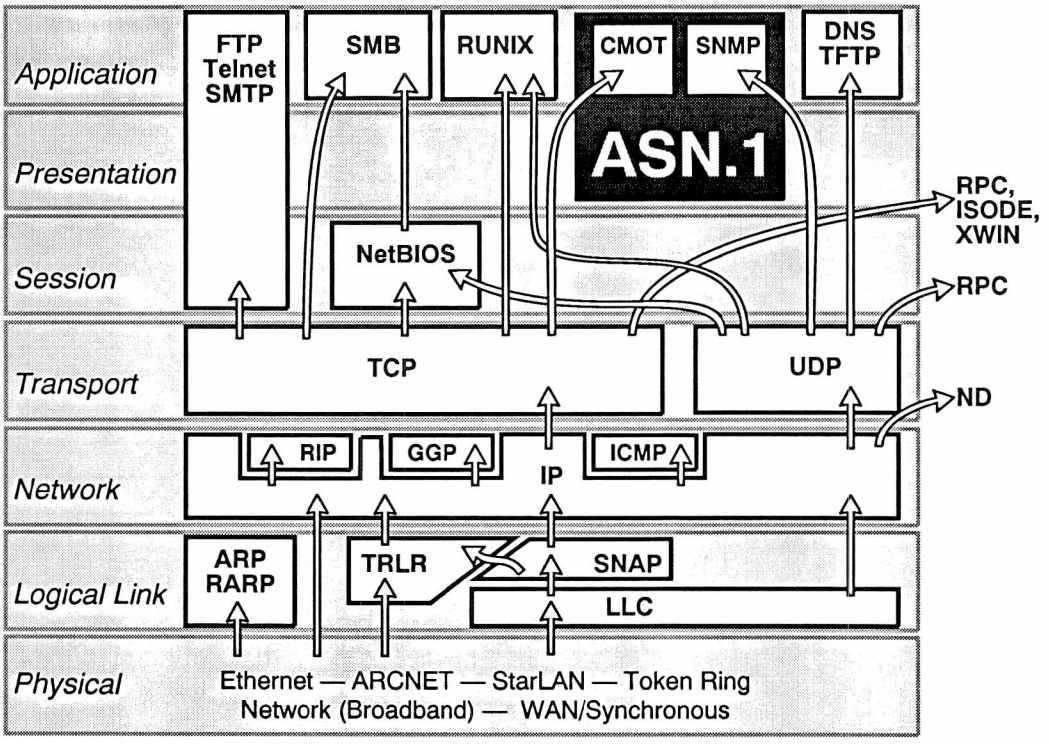

Figure 24. The TCP/IP PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

SMB *Server Message Block.* A family of application-level commands for LAN servers developed by Microsoft for use with the IBM PC LAN Program and frequently used in other environments. Although the IBM PC LAN Program sends SMBs as data within NetBIOS frames, in other contexts they may be sent differently. The TCP/IP PI suite decodes SMBs transported by TCP. The SMB protocol for machines running under OS/2 contains extensions not present in the version for DOS machines. The TCP/IP PI suite decodes both the older DOS versions and the extended OS/2 versions.

NetBIOS *Network Basic I/O System.* A TCP/UDP version of a protocol developed for the IBM PC LAN Program to support communication between symbolically named stations and transfer of arbitrary data. In the TCP/IP context, NetBIOS is over UDP or IP. (TCP/IP implementations of NetBIOS differ from the IBM version, and the NetBIOS module of the TCP/IP PI suite differs accordingly from the corresponding modules of the IBM PI suite and the Novell NetWare PI suite.)

FTP *File Transfer Protocol.* A protocol based on TCP/IP for reliable file transfer.

TFTP *Trivial File Transfer Protocol.* A simple protocol used to exchange files between networked stations with less overhead than FTP.

Telnet Protocol for transmitting character-oriented terminal (keyboard and screen) data.

SMTP *Simple Mail Transfer Protocol.* A protocol for reliable exchange of electronic mail messages.

RUNIX *Remote Unix.* Protocol for handling remote requests to a UNIX™ host, including commands RLOGIN, RWHO, REXEC, RSHELL and remote printing.

DNS *Domain Name Service.* A protocol for finding information about network addresses using a database distributed among different name servers.

TCP *Transmission Control Protocol.* This connection-oriented byte-stream protocol provides reliable end-to-end communication using datagrams sent over IP.

UDP *User Datagram Protocol.* Transmits datagrams over IP.

IP *Internet Protocol.* Handles end-to-end forwarding and long packet fragmentation control.

RIP *Routing Information Protocol.* Exchanges routing information among gateways and end systems.

GGP *Gateway-to-Gateway Protocol.* Exchanges routing information among IP gateways.

ICMP *Internet Control Message Protocol.* Reports on difficulties in datagram transmission.

LLC *Logical Link Control (IEEE 802.2).* Provides connection control and multiplexing to subsequent embedded protocols.

ARP *Address Resolution Protocol.* Finds a node's DLC address from its IP address.

RARP *Reverse ARP.* Finds a node's IP address from its DLC address.

SNAP *Sub-Network Access Protocol.* Also called Sub-Network Access Convergence Protocol.

TRLR *Trailer format.* Variant of IP in which the protocol headers follow rather than precede the user data.

SNMP *Simple Network Management Protocol.*

CMOT *Common Management and Information Services Protocol (CMIP) over TCP.* A management protocol for network; it uses ASN.1 encoding.

SUN® Protocol Interpreter Suite

The Sniffer analyzer interprets the protocols that support Sun Microsystems' Network File System (NFS). NFS allows users at workstations to mount directories of files that are located on other machines and to treat them as if they were locally available through the client's operating system. NFS provides an interface that permits a variety of machines (not necessarily under the same operating system) to play the roles of client or server. NFS is composed of a modified UNIX kernel, a set of library routines, and a collection of utilities used by machines that play the role of server.

The Sun PI suite makes use of the session and transport layers of a host network and does not include lower-level protocols of its own. Typically (but not necessarily) Sun NFS runs over TCP/IP on Ethernet. The SUN PI suite interprets frames passed to it by the TCP, IP or UDP protocols and, thus, requires the TCP/IP PI suite.

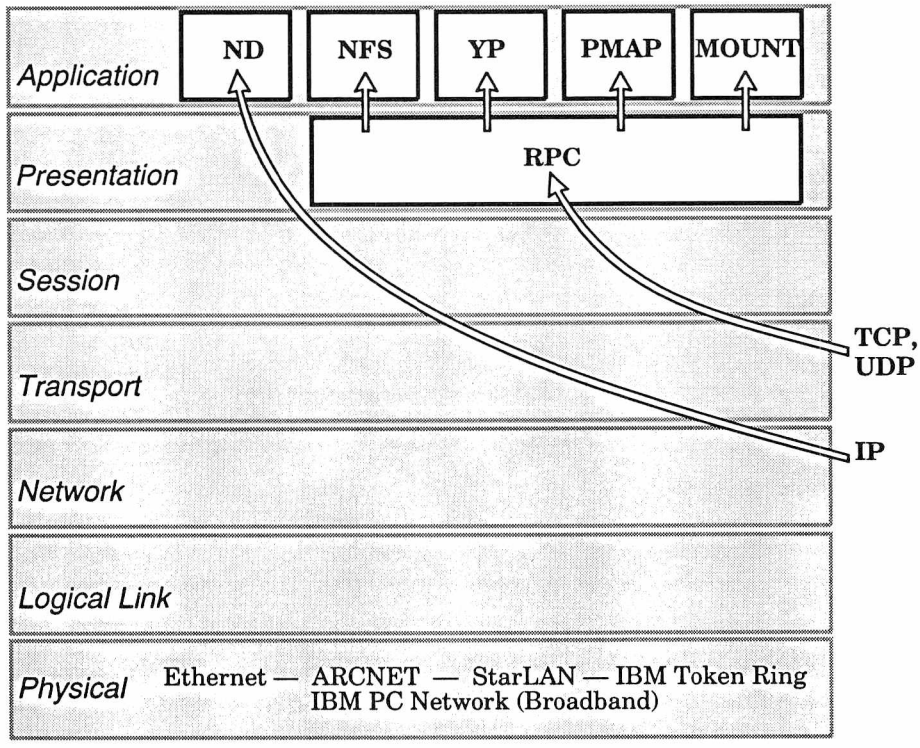

Figure 25. The SUN PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

ND *Network Disk.* A protocol used to access virtual disks located remotely across the network and to boot diskless workstations.

NFS *Network File System.* The high-level protocol used for communication of requests and responses between network clients and NFS servers. The Sun PI suite interprets NFS Version 2.

YP *Yellow Pages.* A high-level protocol used for requests and responses regarding the availability of network hosts, services and directories from a read-only network database.

PMAP *Port Mapper.* A protocol for mapping RPC program numbers to TCP/IP port numbers.

MOUNT A protocol used during initiation of a remote user's access to a network disk, including access checking and account validation.

RPC *Remote Procedure Call.* A protocol for activating a function on a remote station and retrieving the result.

ISO Protocol Interpreter Suite

The Sniffer analyzer interprets the family of protocols built upon recommendations of the International Standards Organization as part of an ongoing international cooperative effort in support of Open Systems Interconnection (OSI). It decodes all layers above the physical layer, which may be any of Ethernet, StarLAN, token ring, IBM PC Network (broadband), or WAN/synchronous. It also decodes Microsoft SMBs.

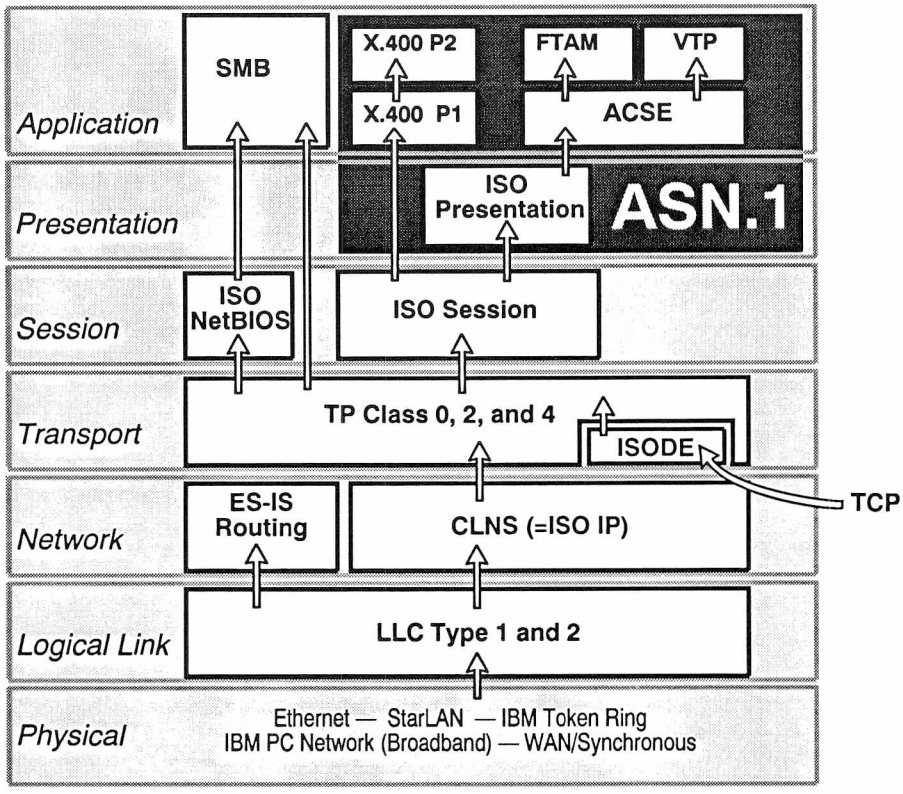

Figure 26. The ISO PI suite shown in reference to the OSI reference model for data communications.

Protocols Interpreted

X.400 The CCITT 1984 protocol for electronic mail. It consists of two levels: P1 for the addressing of the message's outer envelope and P2 for the inner addressing and content of a personal message.

FTAM *File Transfer, Access and Management (ISO 8571/4).*

VTP *Virtual Terminal Protocol (ISO 9041).*

ACSE *Association Control Service Element (ISO 8650/2).* An intermediate application-level protocol used in ISO to support a number of more specific application protocols.

Presentation *(ISO 8823).* Application data is encoded using the basic encoding rules (ISO 8825) of Abstract Syntax Notation One (ASN.1, ISO 8824). The user may choose to interpret part of these messages either by sharing the generic ASN.1 syntax structure or by displaying the semantics specific to the application-level protocol.

Session *(ISO 8327).*

SMB *Server Message Block.* A protocol developed by Microsoft for use with the IBM PC LAN Program to make requests from a user station to a server and receive replies. SMB is part of the protocol family that for DOS machines is called MS NET and for OS/2 machines is called the LAN Manager. The OS/2 version of SMB contains extensions not present in the DOS version; both versions are interpreted in the ISO PI suite.

TP *Transport Protocol (ISO 8073).* The ISO PI suite interprets class 0 (for connection-oriented networks), class 4 (for connection-less networks) and the intermediate class 2.

CLNS *Connectionless Network Service Protocol (ISO 8473).* Also called ISO IP, for Internetwork Protocol.

ES-IS Routing *End-System to Intermediate-System Routing (ISO 9542).* A protocol within the ISO family, used to exchange routing information between gateways and hosts.

LLC *Logical Link Control (ISO 8802/2).*

TCP/IP Frames

ISODE *ISO Development Environment.* A protocol used to encapsulate higher-level ISO messages when they are transmitted over a network whose lower levels use TCP/IP. (ISODE serves primarily as a development technique during transition from TCP/IP to ISO protocols).

DECnet® Protocol Interpreter Suite

The Sniffer analyzer fully decodes eight protocols defined in Phase IV of Digital Equipment Corporation's *Digital Network Architecture* (DNA). It also decodes several additional protocols that, although not specified in DNA, are used in DECnet systems.

DNA was introduced in 1975 as a master plan for a family of networking hardware and software products valid across a range of machines using both wide-area and local-area networks. Implementation is in phases. Current DEC systems implement Phase IV. DEC plans that Phase V will harmonize DEC's architecture with the general OSI model adopted by the ISO.

DEC provides an implementation of DNA phase IV for each of its operating systems. On LANs, DECnet is commonly used by machines whose physical link is by Ethernet or its LAN relatives, such as StarLAN or IBM PC Network (broadband). DECnet protocols can also be used on token ring and WAN/synchronous. The DECnet PI suite can be installed with any of these.

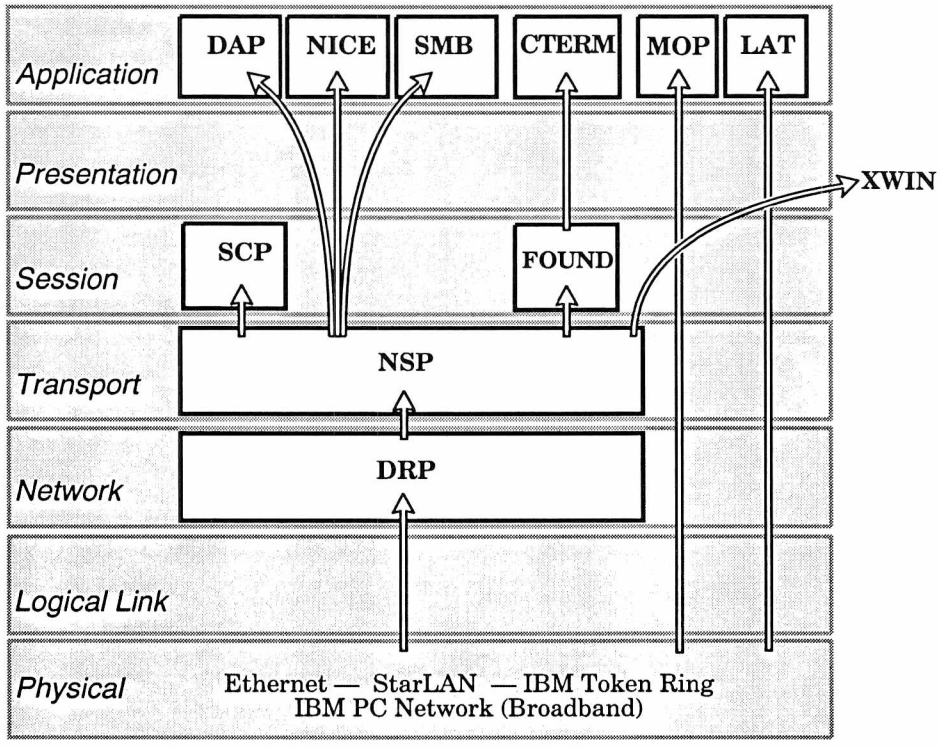

Figure 27. The DECnet PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

DAP *Data Access Protocol.* A protocol that provides remote file access operations. It is a command/response protocol that allows a user process to create new files on a server, open existing files, read and write data, and so on.

NICE *Network Information and Control Exchange.* A command/response protocol that provides network management information.

SMB *Server Message Block.* A message type used by the IBM PC LAN Program to make requests from a user station to a server and to receive replies. Many of the functions are similar to those made by an application program to DOS running on a single computer. It is a protocol for remote file access that is very similar in function to DAP and also dwells in the application layer. It was initially developed for the IBM PC LAN Program and is supported by DEC for compatibility.

CTERM *Command Terminal.* A protocol used for communicating with generic intelligent terminals, i.e., a virtual terminal protocol. It is used in conjunction with FOUND.

FOUND *Foundation Services.* A protocol used for primitive terminal-handling services and to make and break logical connections between applications and terminals. It is used in conjunction with CTERM.

SCP *Session Control Protocol.* A protocol that establishes virtual circuits based on NSP packets.

NSP *Network Services Protocol.* A protocol that provides reliable message transmission over virtual circuits. Its functions include establishing and destroying logical links, error control, flow control, and segmentation and re-assembly of messages.

DRP *DECnet Routing Protocol.* The lowest-level protocol concerned with moving packets from source nodes, through routers, between and within areas, and to end nodes.

MOP *Maintenance Operations Protocol.* A protocol used for network maintenance services that include downline loading, upline dumping, and remote testing and problem diagnosis.

LAT *Local Area Transport Protocol.* A protocol designed to efficiently handle multiplexed terminal (keyboard and screen) traffic to and from timesharing hosts. LAT is a non-DECnet set of protocols that interfaces directly with the LAN and provides an alternative service to CTERM.

Nestar PLAN™ Series Protocol Interpreter Suite

The Sniffer analyzer interprets protocols used with the Nestar PLAN Series of network disk servers. In a network using Nestar servers, each PC workstation, running under DOS, installs a device driver that permits the user to mount and use an arbitrary number of virtual disks. Each such "disk" appears as an additional drive on the local machine but is, in fact, located on a network server. A Nestar server is a dedicated 68000-based machine running under a proprietary operating system which supports a superset of DOS.

In addition to its proprietary commands and their protocols, the Nestar server may also support Microsoft's SMB commands as an alternative access mechanism. The Nestar PLAN Series PI suite interprets them also.

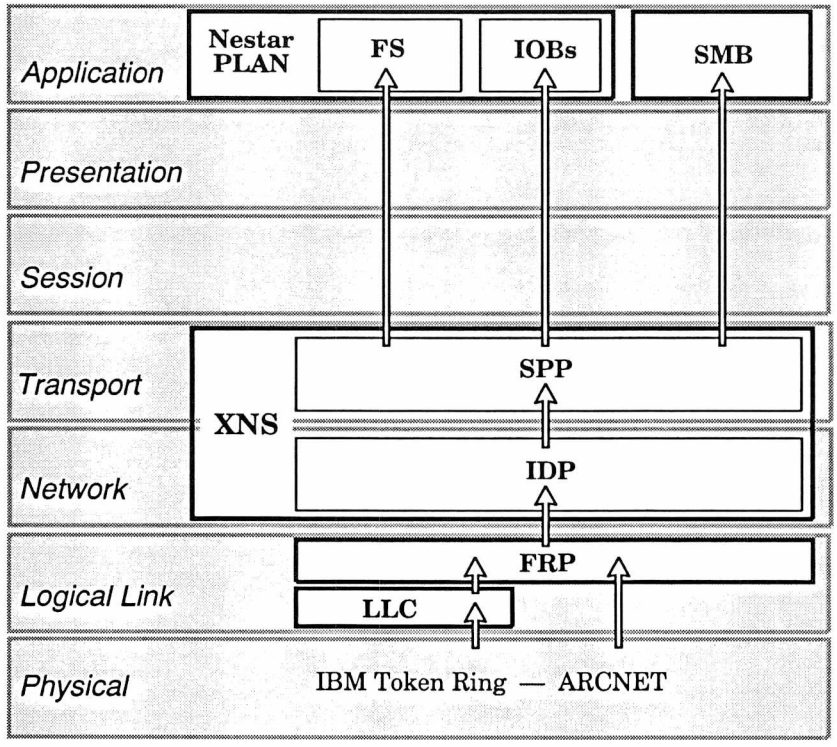

Figure 28. The Nestar PLAN Series PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

Nestar At the application level, Nestar File Server (FS) commands handle user requests to manage the network disks. The repertory includes commands such as CREATE, DELETE, MOUNT, UNMOUNT, LOCK, PROTECT, SHOW, PASSWORD, BINARY LOAD, BINARY SAVE, and others. Unseen to the user and transparent to a user application, the workstation and server exchange Input/Output Block (IOB) messages to support access to files on the network disks.

SMB *Server Message Block.* This is the application level of the protocols used with the IBM PC LAN Program but frequently used in other environments as well. Many of the functions are similar to those made by an application program to DOS or to

OS/2 running on a single computer. Under the IBM PC LAN Program, SMBs are sent as data within NetBIOS frames, but in the Nestar context, they are transported by XNS.

XNS *Xerox Network Systems Protocol.* Within this family of protocols, the Nestar PLAN Series PI suite decodes the network and transport-level protocols, SPP and IDP.

SPP *Sequenced Packet Protocol.* A connection-oriented virtual-call protocol.

IDP *Internet Datagram Protocol.* Delivers to an internet address a single packet as an independent entity, without regard to other packets or to the addressee's response.

FRP *Fragmentation Protocol.* Breaks up and reassembles network-layer packets so that they are acceptable to the data-link protocol and the underlying physical medium.

LLC *Logical Link Control (IEEE 802.2).* Provides connection control and multiplexing to subsequent embedded protocols for devices on the token ring.

Banyan® VINES™ Protocol Interpreter Suite

The Sniffer analyzer interprets protocols in the VINES series developed by Banyan Systems. VINES links personal computers to file servers on a LAN, perhaps with gateway links to other LANs or WANs. The user stations are PCs, typically running under DOS. Redirection permits directories on the servers to appear to the users as DOS drives, although each server in fact offers these services through processes running on the server's Unix operating system. The server role may be played by a wide range of devices from different vendors, but all appear to the user in the same way.

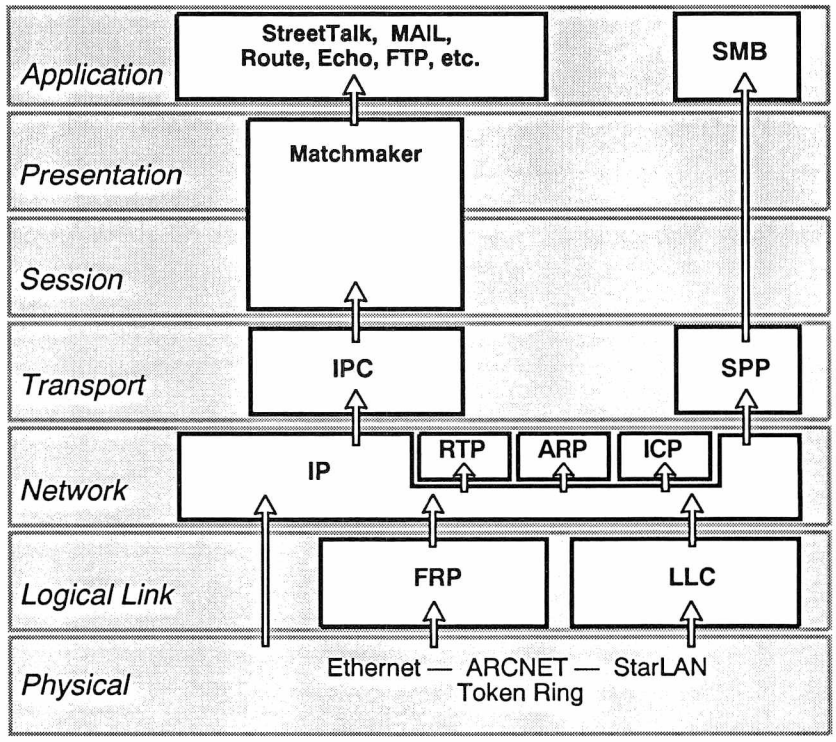

Figure 29. The Banyan VINES PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

StreetTalk Protocol used in Banyan VINES to maintain a distributed directory of the names of network resources. Names are global across the internet and independent of the network topology.

MAIL Protocol for the transmission of messages in the VINES distributed electronic mail system.

SMB *Server Message Block.* A family of application-level commands for LAN servers developed by Microsoft for use with the IBM PC LAN Program but frequently used in other environments as well. Many of the functions are similar to those made by an application program to DOS or to OS/2 running on a single computer. The IBM PC LAN Program sends SMBs as data within NetBIOS frames. In the VINES contexts, they are transported by SPP. The SMB protocol for machines

running under OS/2 contains extensions not present in the version for DOS machines. The Banyan VINES PI suite interprets both the older DOS versions and the extended OS/2 versions.

Matchmaker Protocol used by the VINES service that provides high-level program-to-program communication and remote procedure calls. Matchmaker services include data translation as necessary to match the conventions of sender's and receiver's formats. Matchmaker is descended from the protocol that in XNS is called Courier.

In addition to MAIL and StreetTalk, the Banyan VINES PI suite identifies the following protocols that may be transmitted by a Matchmaker frame: Echo, Router, Background, File, FTP, Sever, Talk, and Network Management.

IPC *Interprocess Communication Protocol.* A transport-level protocol providing reliable message service and unreliable datagram service.

SPP *Sequenced Packet Protocol.* The transport-level protocol to provide virtual connection service, based upon the protocol of the same name in XNS.

RTP *Routing Update Protocol.* Protocol used to distribute network topology information.

ARP *Address Resolution Protocol.* Used for finding a node's DLC addresses from its IP address.

ICP *Internet Control Protocol.* Used to broadcast notification of errors and to note changes in network topology.

IP *Internet Protocol.* The protocol that moves datagrams throughout the network.

FRP *Fragmentation Protocol.* Breaks up and reassembles network-layer packets so that they are acceptable to the data-link protocol and the underlying physical medium and to the IP protocol above it.

LLC *Logical Link Control (IEEE 802.2).* A protocol that provides connection control and multiplexing to subsequent embedded protocols.

AppleTalk® Protocol Interpreter Suite

AppleTalk protocols link personal computers (frequently but not necessarily Apple computers) to each other and to external services such as gateways, file servers, or printers. AppleTalk is commonly used over Apple Computer's LocalTalk, Ethernet, or WAN/synchronous or may be encapsulated within packets transmitted by an unrelated protocol, for example TCP/IP.

The Sniffer analyzer interprets frames in both Phase 1 and Phase 2 of the AppleTalk family of protocols.

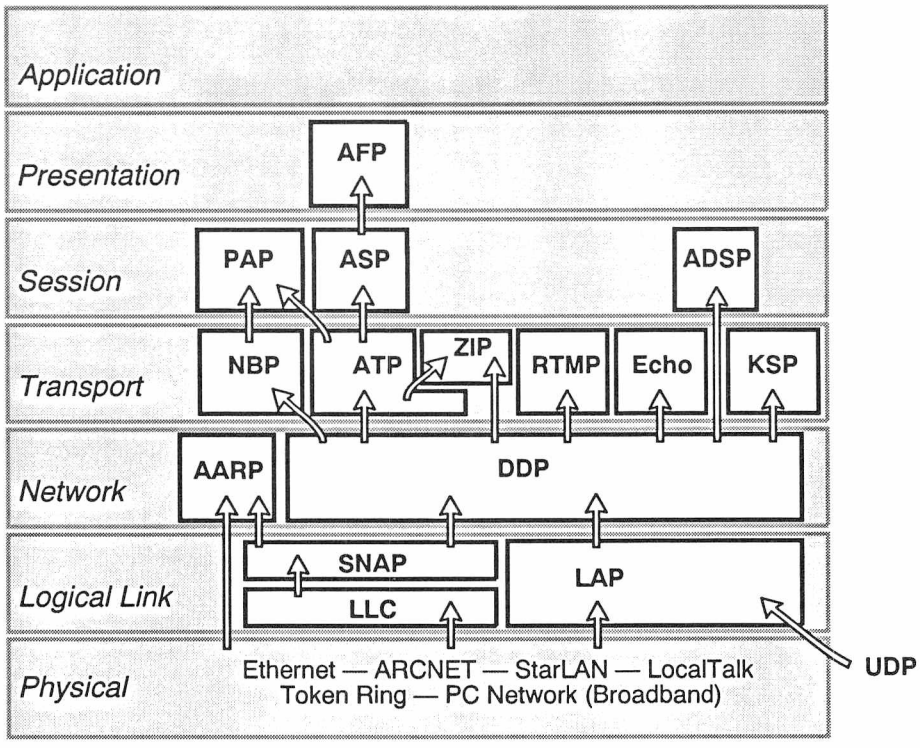

Figure 30. The AppleTalk PI suite shown in reference to the ISO reference model for data communications.

Protocols Interpreted

AFP *AppleTalk Filing Protocol.* A application-level protocol for access to remote files.

PAP *Printer Access Protocol.* A protocol that uses ATP XO ("exactly once") commands to create a stream-like service for communication between user stations and the Apple LaserWriter® or similar stream-based devices.

ASP *AppleTalk Session Protocol.* A general protocol, built upon ATP, providing session establishment, maintenance, and tear-down, along with request sequencing.

ADSP *AppleTalk Data Stream Protocol.* A connection-oriented protocol providing a reliable, full-duplex, byte-stream service between any two sockets on an AppleTalk internet, ensuring in-sequence, duplicate-free delivery of data over its connections.

NBP *Name-Binding Protocol.* Used in AppleTalk networks to permit network users to refer to network services and sockets by character names. NBP translates a character-string name within a zone into the corresponding socket address.

ATP *AppleTalk Transaction Protocol.* Provides a loss-free transaction service between sockets, allowing exchanges between two socket clients in which one client requests the other to perform a particular task and report the result.

RTMP *Routing Table Maintenance Protocol.* Used in AppleTalk networks to allow bridges or internet routers dynamically to discover routes to the various networks of an internet. A node that is not a bridge uses a subset of RTMP (the RTMP stub) to determine the number of the network to which it is connected and the node IDs of bridges on its network.

ZIP *Zone Information Protocol.* Used to maintain an internet-wide mapping of networks to zone names for the benefit of routers and as a resource for the name-binding protocol (NBP) to determine which networks belong to a given zone.

Echo A simple protocol that allows any node to send a datagram to any other node and to receive an echoed copy of that packet in return, to verify the node's existence, or to make round trip delay measurements.

KSP *Kiewit Stream Protocol.* A transport protocol resembling TCP developed at Dartmouth College for the support of terminal emulators.

AARP *AppleTalk Address Resolution Protocol.* Matches the destination address corresponding to a higher-level protocol address.

DDP *Datagram Delivery Protocol.* Extends the services of the underlying LAP protocol to include an internet of interconnected AppleTalk networks, with provision to address packets to sockets within a node.

SNAP *Sub-Network Access Protocol.* Called Sub-Network Access Convergence Protocol.

LLC *Logical Link Control (IEEE 802.2 and ISO/DIS 8802/2).* A protocol that provides connection control and multiplexing to subsequent embedded protocols.

LAP *Link Access Protocol.* The logical-link protocol for AppleTalk. It exists in two variants: ELAP for Ethernet and LLAP for LocalTalk.

X Windows™ Protocol Interpreter Suite

The Sniffer analyzer interprets the protocol used to transmit information between X Windows clients and servers. The protocol is independent of the lower-level frames that carry its messages. The X Windows PI suite must be installed in combination either with the TCP/IP PI suite where it interprets frames passed to it by TCP, or with the DECnet PI suite where it interprets frames passed to it by NSP. DECWindows is Digital Equipment Corporation's name for X Windows over DECnet.

X Windows is an outgrowth of Project Athena at the Massachusetts Institute of Technology in 1984. Its development was supported by contributions from Digital Equipment Corporation and IBM. Development of the X Windows system is now supported by a consortium that includes the original sponsors and more than 40 additional vendors. The X Windows PI suite interprets the protocol of the consortium's current standard, Version 11, Release 4.

The X system permits a task's graphic display to be treated independently of the task itself. An application's computations may be done anywhere— at any mainframe, mini, or micro that is accessible through the network. The display is handled independently for each user by a *display server.* The rest of the application's work is handled by a process that acts as a remote *client* of the end-user's display server. The client does not need to know anything about the server's hardware or software. It simply describes its output in terms of the X interface. The server must turn that description into a display. The server can maintain contact with several clients at the same time and, thus, manage multiple windows, sizing, overlaying, moving or hiding them as the person at the server directs.

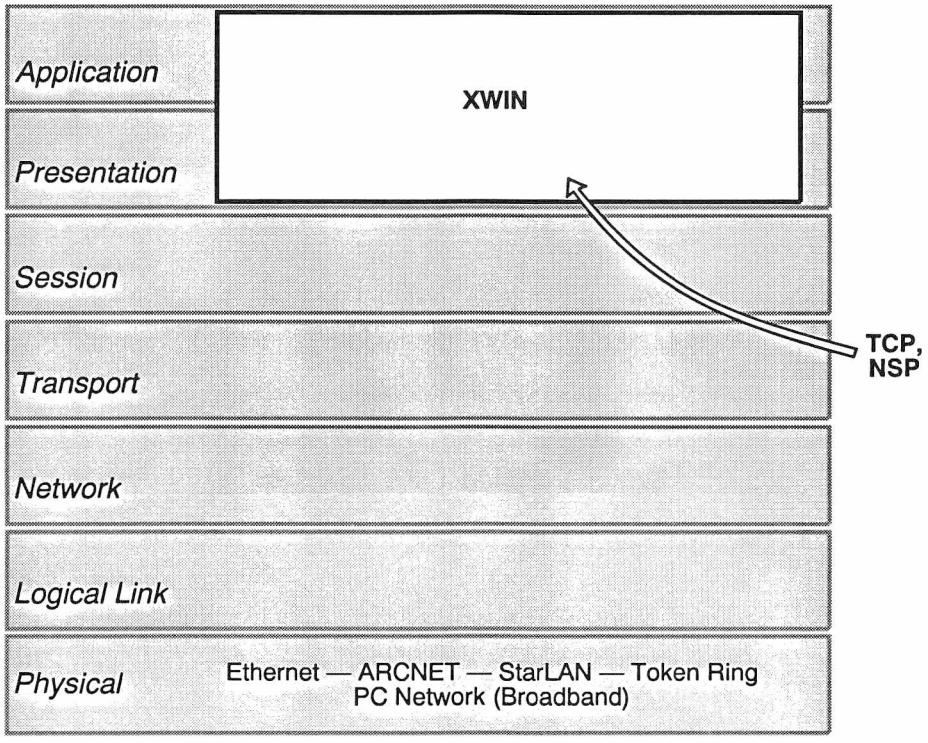

Figure 31. The X Windows PI suite shown in reference to the ISO reference model for data communications.

Features of the Interpreter

The X Windows protocol permits a sequence of several commands to be concatenated in a single X message. For transmission, an X message may be fragmented into several frames. The X Windows PI suite reassembles a fragmented message. The hex and detail displays shows the entire X message starting at the first of its DLC frames. The interpreter's summary window shows a separate line for each X command, regardless of the way the commands may have been packed into lower-level frames.

It may happen that the Sniffer analyzer starts recording after transmission of the initial X Windows setup message. The initial message establishes byte-ordering of the transmitted data, and synchronizes the boundaries of X commands within the transport byte stream. If the interpreter has not seen the initial message, for X messages sent over TCP, the X Windows PI suite uses a heuristic to recognize an X message and to establish the byte-order for its data.

Where a frame includes the selection of options as a sequence of bits, most Sniffer PIs show all the options, as well as an indication of which were selected. However, some X Windows options are so extensive that listing all of them would require dozens or even hundreds of lines. In such cases, the interpreter shows only the options that are selected and omits those that are not.

X.25 Protocol Interpreter Suite

The Sniffer X.25 PI suite fully decodes six protocols used in the communication links of WANs. It decodes the network layer 3 of the standard usually known as Recommendation X.25 of the CCITT. Also, it decodes certain protocols commonly used above X.25, identifies several other higher-level protocols that may be transmitted over X.25, and passes packets to the appropriate PI suites for display.

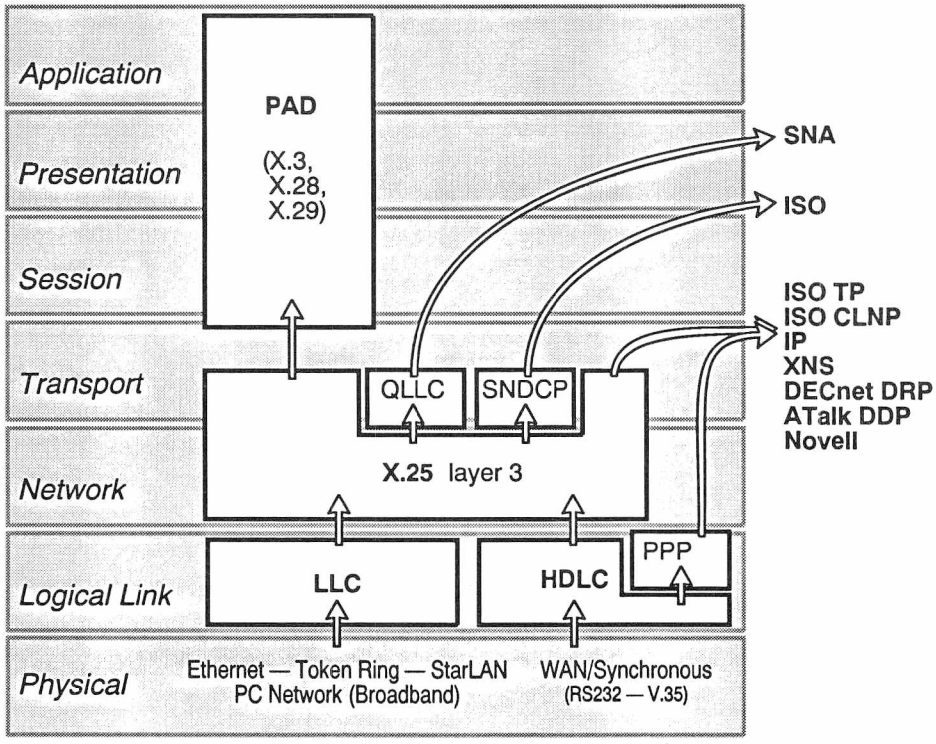

Figure 32. The X.25 PI suite shown in reference to the ISO reference model for data communications.

Features of the Interpreter

PAD *Packet Assembler/Disassembler Protocol.* This protocol family provides buffering between traffic at a terminal or similar character-oriented device and the block-oriented communications of a packet-switched network. CCITT recommendation X.3 describes a virtual device that acts as intermediary between the terminal and the X.25 network. The protocol between the terminal and PAD device is described in X.28 and between the PAD device and the X.25 link in recommendation X.29.

X.25 The Sniffer X.25 PI decodes layer 3 of the 1980 and 1984 versions of CCITT recommendation X.25, including the 1984 extensions for OSI addressing and the ISO and DDN facility and diagnostic fields.

The interpreter recognizes numerous higher-level embedded protocols and (when installed) passes frames to the appropriate PI suite. Protocols thus interpreted include:

ISO TP and ISO CLNP (with the ISO PI suite); IP (with the TCP/IP PI suite); DRP (with the DECnet PI suite); XNS (with the XNS PI suite); DDP (with the AppleTalk PI suite); and NCP (with the Novell NetWare PI suite).

SNDCP *Subnetwork Dependent Convergence Protocol.* This intermediate protocol provides an interface between X.25 and the transport layer of an ISO protocol. (The enclosed ISO protocols are interpreted when the ISO PI suite is also installed.)

QLLC *Qualified Logical Link Control Protocol.* This intermediate protocol provides an interface between X.25 and the SNA family of protocols. (The enclosed SNA protocols are interpreted when the Sniffer IBM protocol interpreter is also installed.)

PPP *Point-to-Point Protocol (RFC 1134).* This link-level protocol by-passes X.25 for communication between systems that are directly connected, running any of a variety of protocols directly over HDLC.

HDLC *High-level Data Link Control Protocol.* This ISO standard protocol is widely implemented as the logical link layer for an X.25 network. (On IBM networks, the corresponding protocol is called SDLC.) The WAN/synchronous Sniffer analyzer interprets LAPB, the subset of HDLC used to provide link-level support for X.25.

3. Extending and Customizing Protocol Interpreters

3

Network General

Chapter 3. Extending and Customizing Protocol Interpreters

Chapter 3. Extending and Customizing Protocol Interpreters

Chapter Overview

Chapter 3 describes the rules and conventions for writing programs that extend the Sniffer analyzer's ability to interpret protocols.

Network General Corporation is constantly expanding its suite of optional PIs. Before writing your own, you should check with NGC to see what is currently available.

To make use of this chapter, you need to be familiar with:

- The general operation of the Sniffer analyzer.
- The general structure of network frames and the detailed structure of the protocol to be interpreted.
- The C programming language and the MS-DOS programming environment.

Several of the existing interpreters have *stub routines* that are called when opcodes or type fields are encountered for which we have no information. To support a protocol that is an extension to an existing protocol, it is much easier to replace or modify the stub routine than to write a new PI.

What Does a Protocol Interpreter Do?

A PI is a routine (or set of routines) that, when invoked, receives a pointer to data somewhere within a frame. The interpreter has four tasks:

- To generate one or more short text lines for its protocol level to be displayed in the **summary** view.
- To generate lines for the **detail** view.
- Call the PIs for embedded protocols, if any.
- Supply new symbolic names discovered within the protocol's field data.

There are two types of PIs: *demultiplexed* and *embedded*. A demultiplexed PI is called directly from the Sniffer analyzer, based on the frame's identifying code. (Depending on the network, that may be its Ethertype, its ARCNET system code, its 802.2 LLC DSAP, and so on.)

An embedded PI is called by another PI to interpret a protocol nested at a higher level. An embedded PI may be shared. That is, a PI may be called from several other PIs and, thus, may be used at different protocol levels.

Calling Conventions for Protocol Interpreters

Suppose you write a PI function named `new_pi`. It is called from the Sniffer analyzer as follows:

```
int new_pi (frame_ptr, frame_length)
char *frame_ptr;                    Pointer to frame data
int frame_length;                   The length of the frame data
                                    starting at frame_ptr
```

The pointer to the frame data starts within the physical frame at the beginning of the data for the protocol. Thus, it skips previous protocol fields, including source and destination addresses, type fields, and any previously-embedded protocol data.

The Sniffer analyzer permits the operator to request that the Sniffer analyzer truncate frames as it captures them. Then the Sniffer analyzer records no more than a certain number of bytes for each frame and discards the rest. When the operator has elected truncation, `frame_length` is the length of the stored (truncated) data, and the global integer `bytes_not_present` indicates how much additional frame data was received but not recorded.

The following rule applies only to Ethernet, StarLAN, and token ring. For a demultiplexed PI called for a particular LLC DSAP, `frame_ptr` is the address of the first byte of the information field. The information field immediately follows the source and destination SAPs and the control field. The SAP protocol interpreter is called for any frame with an information field, such as `I`, `UI`, or `XID`. It is not called for frames that do not contain information, such as `RR` or that do not contain higher-level protocol information, such as `TEST` or `FRMR`.

The following rule applies only to Ethernet, StarLAN, and PC Network. For a demultiplexed PI called for a particular Ethertype, `frame_ptr` is the address of the first byte of the information field. The information field immediately follows the 2-byte Ethertype.

A PI's return value is the number of bytes of frame data that it interpreted. The calling interpreter may use the return value to decide whether any subsequent interpreters are to be called. If there is no more data left to interpret, simply return the initial value of `frame_length`.

If the PI needs access to DLC or LLC header fields, or to the frame number, it may refer to the values of certain global static variables in which these are recorded. They are listed in Figure 33.

Global Static Variable		Purpose
`long`	`pi_frame;`	The current frame number.
`char`	`*dlc_header;`	Pointer to DLC header starting with the first byte of the frame.
The following are used only for 802.3 networks, Ethernet, StarLAN, and token ring:		
`char`	`*llc_header;`	A pointer to the LLC header of the frame, starting with the DSAP field.
`int`	`llc_type;`	The type of the LLC frame; see the `LLC_xxx` macros in `pi.h`.

Figure 33. Global static variables available to the protocol interpreter.

A PI for an embedded protocol should be invoked using the same calling convention. It is possible to include additional parameters if necessary.

Registering Protocol Interpreters

Each PI should be registered before it is called. When you have registered a protocol, its name appears in the list of protocols available for selection in the **display filter** menu, so the user can select frames that contain your protocol.

For a demultiplexed PI, registration is required, so that the Sniffer analyzer knows about it. For an embedded PI, registration is optional.

All registration occurs in the function `initpi.c`, called when the Sniffer analyzer is initialized. Registering a PI generates a pointer to a structure that holds data relevant to the interpreter. That pointer must be supplied in subsequent calls that generate screen output. The pointer should be saved in a static variable known to the interpreter.

The function that registers a PI has the following calling sequence:

```
struct pi_data  *register_pi (menu_title, pi_type, ntypes, types, new_pi, prefix)
```

`char *menu_title;` — A pointer to the string that will appear in the display filter menu as a selectable item and which the operator can use to control whether **summary** lines for that protocol are displayed. The string is also used in constructing the message that appears at the bottom of the menu panel. The string should not exceed 18 characters.

`int pi_type;` — The type of PI; see the `PITYPE_xxx` symbols `pi.h`. You can also choose a color to be used for the **summary**, **detail**, and **hex** views from your PI by adding one of the color symbols to `pi_type`. Color symbols are defined as macros in `initpi.c` and should generally be chosen to correspond to the OSI model network level of your protocol. For example, an embedded application level might specify `PITYPE_EMBEDDED + L7`. If the color is omitted, white will be used.

The most common instances of `PITYPE` variables are as follows:

ARCNET:	`PITYPE_ARCID`	A system-code-demultiplexed PI.
Token ring, Ethernet, StarLAN and PC Network:	`PITYPE_SAP`	A SAP-demultiplexed PI.
Ethernet, StarLAN or PC Network:	`PITYPE_ETYPE`	An Ethertype-demultiplexed PI.
Any:	`PITYPE_EMBEDDED`	An embedded PI called by another.

`int ntypes;`

The number of types that can be processed by this demultiplexed interpreter. For an embedded protocol, specify 0. Type depends upon the network as follows:

ARCNET:	`ntypes`	The number of system codes
Token ring:	`ntypes`	The number of DSAPs
Ethernet, StarLAN and PC Network:	`ntypes`	The number of DSAPs or Ethertypes

`int *types;`

An array of "ntypes" integers representing the various system codes, DSAPS, or Ethertypes (as appropriate to the network) that can be processed by this demultiplexed interpreter.
For an embedded protocol, this parameter is ignored and should be specified as a null address.

`int *new_pi();`

The address of the PI function that is called as previously described.

`char *prefix;`

A pointer to a string containing an abbreviation identifying this PI. It appears as a prefix for lines in the **detail** view.

To preserve alignment with displays produced by the Sniffer analyzer's other protocol interpreters, the string should be in the form "`xxxx:`" Use three to five, the last one of which should be a colon.

The Protocol Interpreter Data Structure

This structure is allocated, and a pointer to it returned, by the `register_pi()` function. Only the fields described below are relevant to PIs. They are booleans that indicate what functions are required. The integer booleans, in standard C fashion, are zero if false and non-zero if true. The declaration for this structure is part of the file `pi.h`.

```
struct  pi_data {
    int     do_sum;     Boolean: generate summary lines?
    int     do_int;     Boolean: generate interpretation lines?
    int     do_count;   Boolean: only count summary lines?
    int     do_names;   Boolean: add symbolic station names?
```

`int recursive;`	Boolean: recursive call to get information for another frame?

`};`

If `do_sum` is true, you are to generate a **summary** line. If `do_count` is also true, then only a count of **summary** lines is really needed, so for added efficiency you may optionally allocate the **summary** line buffer but omit actually generating the text.

If `do_int` is true, you are to generate one or more **detail** interpretation lines. The operator cannot elect options that would make the `do_sum` and `do_int` flags true at the same time.

If `do_names` is true, the operator has selected the **Search for names** menu option. Examine the frame data for embedded station names defined by the protocol. If any are found, call the `add_station_name()` function, described later, to enter the names into the name table.

If `recursive` is true, this is a call from another PI, seeking information about some other frame. If you do not have the required information, do nothing except call any more deeply-embedded PIs that you know about. (See the section, "Advanced Topic: Dependencies on Other Frames.")

Regardless of how the flags are set, always call the corresponding PI for each protocol embedded in the current one.

Generating Output from Protocol Interpreters

To generate a line for the **summary** view from within a PI, first get the address of a line buffer by calling `get_sum_line()`:

`char *get_sum_line (pid)`	Returns a pointer to the line buffer
`struct pi_data *pid;`	The value returned when the interpreter was registered

Then move a character string (ending with a null) into the buffer provided. The length of the string including the null cannot exceed `MAX_SUM_LINE`. (For visual consistency of the displayed output, the string should begin with a 3-character identification of the protocol layer, a colon, and a blank.)

Generating a line for the **detail** view is similar. However, the function to get the address of a line buffer also provides an optional offset and length that show where in the frame the information was found. This is used to highlight the **hex** field when the operator selects the corresponding line of the **detail** view.

`char *get_int_line (pid, offset, length)`	Returns a pointer to the line buffer
`struct pi_data *pid;`	The value returned when the interpreter was registered.
`int offset;`	The offset from the DLC header of the field that generated the interpretation line.
`int length;`	The length of the field, or 0 if no highlighting is desired.

The string should be built in the supplied buffer. Its length, including the final null, must not exceed `MAX_INT_LINE`.

In general, the Sniffer analyzer automatically scrolls the **detail** view so that the top line is the first line for the protocol that the operator selected in the **summary** view. If you wish some other line of your protocol's **detail** lines to scroll to the top, you may so indicate by using a negative offset when you call `get_int_line` for that line.

Generate **summary** or **detail** display lines only when the appropriate Boolean in the `pi_data` structure is true. Regardless of how the flags are set, always call the PIs for any protocols embedded in the current one.

Adding Symbolic Names to the Name Table

When your PI is called and `do_names` is true, you must examine the frame for an embedded name. If you find one, add it and its symbolic equivalent to the station name table. When you discover a useful name, call the function `add_station_name`, as follows:

```
add_station_name (flagstype, length, addr, name)
```

`int flagstype;`	The type of the address plus option flags; see `pi.h` for definitions. *Example:* `ADDRTYPE_DLC + ASN_NOREPL` indicates that the address is a data link-level address and that its symbolic name is not to replace a symbolic equivalent already assigned to that address in the name table.
`int length;`	The length of the address, from 1 to 16 bytes. *Example:* 6 for datalink addresses.
`char *address;`	A pointer to the binary station address for this name.
`char *name;`	A pointer to the ASCII name to enter in the table; 1 to 31 characters (including a final NUL).

Since names discovered in the protocol data may be transitory, it is probably best to use `ASN_NOREPL`. That way, when the names file already supplied a symbolic equivalent for the address, the name you find does not replace it.

Declaring Embedded Addresses

Each frame has two DLC-level addresses associated with it by the Sniffer analyzer. If a PI knows of other kinds of addresses that are associated with this frame, such as Internet addresses embedded within the frame data, it may announce those addresses by calling the following routine:

```
add_frame_addr (flagstype, length, addr)
```

`int flagstype;`	The type of the address plus `AFA_xxx` option flags; see `pi.h` for definitions. *Example:* `ADDRTYPE_IP + AFA_SRC` for a TCP/IP internet source address of this frame.

`int length;`	The length of the address, from 1 to 16 bytes. Example: 4 for TCP/IP internet addresses.
`int *address;`	A pointer to the binary station address to be added.

Displaying Symbolic Names

If you wish to include the symbolic name corresponding to an address at any level in the text of a **summary** or **detail** line, you may use the following routine to look up and format names from the current name table:

```
char *format_addr (line, length, addr, flagstype)
```

`char *line;`	The address of a character buffer into which the formatted name is placed. The function return value is the address of the null at the end of the string.
`int length;`	The length of the address, from 1 to 16 bytes. Example: 6 for DLC-level addresses.
`char *address;`	A pointer to the binary station address to be looked up in the name table.
`int flagstype;`	The type of the address plus `FMT_xxx` option flags; see `pi.h` for definitions. Example: `ADDRTYPE_DLC + FMT_BOTH` for a DLC address to be formatted with both the hex value and the symbolic name.

Adding Summary Line Flags

If you want to create special flags and to have the Sniffer analyzer flag frames and display the flags in the **summary** view, you can call the following function:

```
void add_frame_flags(str)
```

`char *str;`	Characters to add to the flags

At most, six flag characters can be displayed for each frame. (See the section, "Flags Option," in the *Sniffer Network Analyzer Operations Manual.*)

Using Other Protocol Interpreters

You may also want to refer to the section, "Augmenting Existing Protocol Interpreters."

In most cases, the PI that you write will be at the top of an already-interpreted protocol stack, and the only other PIs you call will be your own. You may, however, also call existing PIs if the protocol they interpret is embedded within yours. See the `initpi.c` file for a complete list of the PIs that are available, but remember that only those that are part of the installed PIs will be in your library. Also note that new PIs are constantly being added. Contact the factory for the latest list and announcements of new PIs.

The mechanisms by which PIs decide what other PIs to call are varied. The best protocols, from the PI writer's point of view, are those that contain a universally-assigned identifier–such as a well-known port number–that indicates what protocol is at the next level. Without such an identifier, the PI must rely on other

context, sometimes from previous frames, to determine what the protocol stack is for each frame. The PIs that come with the Sniffer analyzer use a variety of techniques, from maintaining session-connection tables across many frames to heuristic identification of likely candidate protocols. A complete description of these techniques is beyond the scope of this manual.

The resulting dynamic nesting of PIs can be complex, and a given PI may be called from several alternative protocol stacks. The following diagrams show the structure of a few of the PIs. Note that `interp_smb`, the interpreter for PC LAN Program SMBs, occurs in several places.

```
interp_dlc

    IF 802.2 network THEN
        IF SSAP == DSAP = 0xFF THEN
            interp_xns (Novell)
                interp_smb
                    interp_smb_other

        ELSE interp_llc
            SWITCH on DSAP

            SAP AA: interp_snap
                        IF vendor == 0 THEN {same as ETHERTYPE below}
            SAP FE: interp_isonw
                        interp_isotp
                            interp_smb
                                interp_smb_other

    ELSE IF Ethernet/Ethertype network THEN
        SWITCH on ETHERTYPE

        0200: interp_pup

        0600, 0807: interp_xns
                        interp_smb
                            interp_smb_other
        0800: interp_ip
                    SWITCH on protocol
                     1: interp_icmp
                     6: interp_tcp
                            SWITCH on port
                            21: interp_ftp
                            23: interp_telnet
                            25: interp_smtp
                    17: interp_udp
                            SWITCH on port
                            53:   interp_domain
                            111:  interp_rpc (Sun)
                            2049: interp_rpc (Sun)
                                        SWITCH on program number
                                            100000: interp_pmap
                                            100003: interp_nfs
                                            100004: interp_yp
                                            100005: interp_mount
        0806, 8035: interp_arp

        6001: interp_decmopdl
        6002: interp_decmoprc
        6003: interp_decdrp
                    interp_decnsp
                        interp_decdap
                        interp_decscp
                        interp_decnice
        6004: interp_declat

        9000: interp_loop

        9002: interp_bridge
```

Figure 34. The structure of the PI for networks using Ethertypes (Ethernet, StarLAN, PC Network).

```
interp_dlc
    SWITCH on type
    00: interp_mac

    01: interp_ri
            interp_llc
                SWITCH on DSAP
                    SAP 04: interp_sna
                    SAP 05: interp_sna
                    SAP 08: interp_sna
                    SAP 0C: interp_sna

                    SAP 80: interp_xns_sap
                              SWITCH on first word:
                              0600: interp_3com
                                        interp_xns
                                            if SPP then interp_smb
                                                interp_smb_other

                              other:interp_xns
                                       if SPP then interp_smb
                                            interp_smb_other
                                   interp_bridge

                    SAP 86: interp_narc
                               interp_nestar
                                   interp_xns
                                       if SPP then interp_smb
                                           interp_smb_other

                    SAP AA: interp_snap
                             IF vendor == 0 THEN SWITCH on ethertype
                             0200: interp_pup
                             0600: interp_xns
                             0800: interp_ip
                                 SWITCH on protocol
                                  1: interp_icmp
                                  6: interp_tcp
                                      SWITCH on port
                                      21: interp_ftp
                                      23: interp_telnet
                                      25: interp_smtp
                                 17: interp_udp
                                      SWITCH on port
                                      53:   interp_domain
                                      111:  interp_rpc (Sun)
                                      2049: interp_rpc (Sun)
                                             SWITCH on program number
                                                   100000: interp_pmap
                                                   100003: interp_nfs
                                                   100004: interp_yp
                                                   100005: interp_mount
                             0806: interp_arp

                             6001: interp_decmopdl
                             6002: interp_decmoprc
                             6003: interp_decdrp
                                       interp_decnsp
                                           interp_decdap
                                           interp_decscp
                                           interp_decnice
```

Continued on next page...

Figure 35. The structure of the PI for networks using token ring.

...continued from previous page.

```
                    6004: interp_declat
                    8035: interp_arp (and rarp)
                    8006: interp_nestar
                    9000: interp_loop

          SAP E0: interp_novell
                      interp_xns
                          if SPP then interp_smb
                              interp_smb_other

          SAP F0: interp_netbios
                      interp_smb
                          interp_smb_other
                      interp_netbios_other

          SAP FA: interp_ungbass
                      interp_xns
                          if SPP then interp_smb
                              interp_smb_other

          SAP FE: interp_isonw
                      interp_isotp
                          interp_smb
                              interp_smb_other
```

Figure 35. Continued.

Figures 34 and 35 above show only a subset of the PIs and how they are interrelated; the true structure is considerably more complex.

Some of the existing PIs call null stubs to process command codes or types that they do not recognize. By replacing or modifying those stubs (whose C-language source is supplied), you can extend those interpreters without rewriting them.

Currently the NetBIOS, SMB, and X Windows interpreters call null stubs. They call routines in the following files to process unknown message types: `intnetbo.c`, `intsmbo.c`, and `xwinext.c`. The calling convention is the same as for registered PIs. In fact, you may register your extension of those stubs so that they can be independently selected as a display filter.

You can modify two tables to call your new PI. The file, `tcpports.c`, contains demultiplexing tables for protocols in the TCP/IP family and for adding IP, TCP, and UDP interpreters on well-known port numbers. The file, `netmgmt.c`, contains tables for *management information data bases* (MIBs) that can be interpreted by the Simple Network Management Protocol (SNMP) in the TCP/IP PI suite. You can find instructions for modifying these tables in comments in the code.

Another approach to extending existing PIs, when there is no stub routine or it is not called at the right time, is to write a new PI that acts as a capture filter to the existing PI. In this case you can piggyback on the existing registration of the PI by changing the name of the function in the module `initpi.c` to be the name of your new routine. When your routine gets control, it looks at the frame data passed to it to see if it is the extension to the protocol that it knows how to handle. If so, it interprets the data and returns. If not, it calls the existing PI under its original name.

Note that to use any of these extension techniques, it is not necessary to have the source code for the existing PIs.

Advanced Topic: Dependencies on Other Frames

In cases where the interpretation of a frame must depend on information in prior frames, there are three mechanisms available. In increasing order of complexity they are:

- The PI can cache any information that will be useful for subsequent frame interpretation in private static variables. Beware, however, that the PI may be called to interpret frames in any order. The cache, therefore, will only sometimes be valid.

  In order to recognize that cached data is invalid when new frames have been loaded or captured, the PI can examine the global integer variable `data_version`, that is incremented each time new data is present.

  This is the easiest and most efficient of these frame dependency mechanisms; in many cases it is sufficient.

- The PI can get access to the data in other frames by calling the function `pi_get_frame`. Its return value is the size of the frame, provided the length is available. The return is zero when the frame does not exist, or, on token ring, when it is not an LC frame.

```
int   pi_get_frame (frame_num, p_dlc, p_llc, p_data)
```

`int`	`frame_num;`	The number of the frame you want. The first frame is frame number 0.
`char`	`**p_dlc;`	The address of the pointer that the function will set to the start of the DLC header.
`char`	`**l_llc;`	The address of the pointer that the function will set to the start of the LLC header for token ring networks.
`char`	`**p_data;`	The address of the pointer that the function will set to the start of the data following the LLC header for token ring networks.

Note that this technique requires the PI to know the position of the data relevant to it, stated as the offset from the start of the data portion of the frame. This means that the PI for an embedded protocol must take into account the structure of the protocol in which it is embedded.

- The PI can request that the PIs be invoked for previous frames by calling the following routine:

```
int   pi_invoke_pis (frame_num)
```

`int`	`frame_num;`	The desired frame number. The first frame is frame number 0.

The return value is non-zero when the frame number is valid and the PI is successfully called.

In general this causes PIs to be invoked recursively. They are informed that this is a recursive invocation by the `recursive` flag in their data structure. If they have

information that is needed by the originating interpreter, information may be exchanged by using static shared variables. For example, the recursively invoked interpreter might check whether it is the request frame for the currently active response, and if so indicate what the original request was.

A useful technique for increasing efficiency when frames must be recursively examined is to completely scan all frames once when new data is captured or loaded, rather than to search back each time a frame is interpreted. As the scan proceeds you can keep information from earlier frames (such as connection identifiers and port numbers) in a temporary table, and refer to that information to determine how to interpret other frames.

As a result of the complete scan you can save what is needed for subsequent interpretation of each frame in a private static data area; this can often be a simple "protocol type" byte. If the data needed for each frame is small, you may use a storage area provided for each frame. When you are called to interpret a frame, the variable `pi_frame_data` (declared in pi.h) will point to an area of `PI_FRAME_DATA_SIZE` bytes (currently three) that is unique to each frame.

Special care must be taken if it is possible that more than one interpreter in the protocol stack for a single frame might be making recursive calls. First, the interpreters must agree by convention about the use of the `pi_frame_data` storage area. Second, each interpreter must distinguish its recursive calls from those of other interpreters.

The following code fragment shows the structure of an interpreter that does an initial frame scan, uses the `frame_data` area, but can otherwise coexist with other interpreters that do recursive calls:

```
interp_xxx (pointer, length)
char *pointer;
unsigned length;

{
static int old_data_version = -1;
static boolean our_scan = FALSE;

    if (!pi_data_xxx->recursive
     && data_version != old_data_version) {         /* do our scan */
        int frame;

        our_scan = TRUE;
        old_data_version = data_version;
        frame = 0;
        while (pi_invoke_pis (frame++))
            ;
        our_scan = FALSE;
        }

    else if (our_scan) {    /* we have been called during the scan */
             /* compute what will be needed to display this frame */
        *pi_frame_data = ; /* store it in the frame_data area     */
        }


    if (pi_data_xxx->do_sum) {          /* generate a summary line */
        }

    if (pi_data_xxx->do_int) {          /* generate detail lines   */
        }

    /*  call embedded protocol interpreters here...                 */
      interp_yyy (...);

    return length;
    }
```

This is the least efficient of the three mechanisms, but it is the most general and is entirely independent of earlier protocol headers. This is especially important when writing embedded PIs that may be invoked from more than one protocol stack or when earlier headers are variable in size.

Debugging Messages

To write a debugging message from within a PI, use the function `debug_msg ("Message of up to 100 characters", optional_variables)`. It is similar to `printf()` in that the string to control formatting may apply to variables included as the subsequent arguments. While it is running, the Sniffer analyzer saves the last 100 such messages in a scrollable window that can be made visible by pressing **Shift-F1** and can be closed by using the **Esc** key. **Shift-F1** will do nothing if no messages have been written to the debugging window.

Advanced Topic: Using the Protocol Interpreter Formatting Routines

PIs may be written with the use of no additional functions other than those already described and those available in the standard C library. For fixed-format data, the

simplest approach is often to write structure declarations and simply format **summary** and **detail** lines using the standard `sprintf()` library function.

When the data contains many variable-length or optional fields, however, using C-language structures becomes quite awkward. To address this problem, we provide a series of stream-oriented formatting utilities called *protocol interpreter formatting* (PIF) routines. The use of PIF routines is entirely optional. You may write interpreters without them if you choose.

A PI that wishes to use the PIF routines makes one call to the initialization function `pif_init()` each time it is called for a new frame. The PIF routines then maintain an offset into the frame data, called the *PIF offset*, that is used to extract data in various forms. There are three general classes of routines:

- Routines that return a data item to the caller begin with `pif_get_`. The PIF offset is not updated. These routines may be used to extract data for either the **summary** line or the **detail** view.
- Routines that display a data item on a line in the **detail** view begin with `pif_show_`. The PIF offset is incremented by the length of the data item and thus points to the next item.
- Other miscellaneous PIF functions.

The PIF Routines

Figure 36 presents a summary of the PIF routines that are available. All three categories presented above are represented.

pif_init	Initialize PIF global variables.
pif_save	Save PIF info before calling an embedded PI.
pif_restore	Restore PIF info after calling an embedded PI.
pif_get_byte	Get value of a single byte.
pif_get_word	Get value of 2-byte word in low-high order.
pif_get_word_hl	Get value of 2-byte word in high-low order.
pif_get_long	Get value of 4-byte longword in low-high order.
pif_get_long_hl	Get value of 4-byte longword in high-low order.
pif_get_ascii	Get ASCII characters into a C string.
pif_get_ebcdic	Get EBCDIC characters into a C string.
pif_get_lstring	Get a length/string into a C string.
pif_get_addr	Get the address of the current field.
pif_show_byte	Display a single byte.
pif_show_word	Display 2-byte word in low-high order.
pif_show_word_hl	Display 2-byte word in high-low order.
pif_show_long	Display 4-byte longword in low-high order.
pif_show_long_hl	Display 4-byte longword in high-low order.
pif_show_2byte	Display 2 one-byte fields.
pif_show_4byte	Display 4 one-byte fields.
pif_show_6byte	Display 6 one-byte fields.
pif_show_nbytes_hex	Display an n-byte field in hexadecimal.
pif_show_ascii	Display a string of ASCII characters.
pif_show_ebcdic	Display a string of EBCDIC characters.
pif_show_lstring	Display a length/string of ASCII characters.
pif_show_flag	Initialize to display bits of a flag byte.
pif_show_flagbit	Display flag bit values.
pif_show_flagmask	Display flag bit value if it matches the mask.
pif_show_date	Display a date and a time.
pif_show_space	Display a blank line
pif_header	Display a **detail** view header message.
pif_trailer	Display a **detail** view trailer message.
pif_autoscroll	Set **detail** view autoscroll point to be the next header.
pif_line	Return a **detail** view line buffer and advance the pointer.
pif_set	Set current buffer pointer.
pif_skip	Move current buffer pointer backwards or forwards.

Figure 36. List of PIF routines.

Summary of PIF Routines

The boxes that follow contain descriptions of the calling sequences presented in Figure 36.

```
void    pif_init (pi, p, len)    /* Initialize PIF globals                    */
struct  pi_data *pi;             /* Protocol interpreter's control block ptr */
char    *p;                      /* Start of frame data to interpret         */
int     len;                     /* Length of frame data                     */
```

Initialize the PIF variables. `PIF_INIT` must be called by the PI routine before any other `PIF_xxx` routines can be used.

```
void    pif_save (pd)            /* save pif variables                       */
struct  pif_info *pd;            /* Ptr to area used to save pif state       */
```

Save the PIF variables before calling an embedded PI. The caller supplies a "`pif_info`" area (defined in `pi.h`) into which the current state information is saved. The `pif_restore` routine can be used to restore the state information. This should be used before calling an embedded protocol interpreter that uses the PIF routines if you wish to use the PIF routines after it returns.

```
void    pif_restore (pd)         /* restore pif variables                    */
struct  pif_info *pd;            /* Ptr to area used to restore pif state    */
```

Restore the PIF variables. The caller supplied a "`pif_info`" area that was previously supplied to `pif_save`.

```
char    pif_get_byte (delta)
int     pif_get_word (delta)
int     pif_get_word_hl (delta)
long    pif_get_long (delta)
long    pif_get_long_hl (delta)
int     delta;
```

Fetch a byte, word (2 bytes) or longword (4 bytes) from the frame data at the current PIF offset plus the (signed) value of delta. The PIF offset is not changed. The "`_hl`"versions are for multibyte fields stored with the most significant byte first. These are macros defined in `pi.h`.

```
char   *pif_get_ascii (offset, len, result_str)   /* get asciiz string      */
int    offset;                 /* offset to string from current pif pointer */
int    len;                    /* maximum number of source bytes            */
char   result_str[];           /* destination string                        */
```

Move a printable ASCII zero-terminated string at the current offset in the packet to a C string. Unprintable characters are replaced with the character "`.`". The destination string ends with a "`\0`"even when the source doesn't. The return value is a pointer to the *end* (null) of the destination string. The PIF offset is not changed.

```
char    *pif_get_ebcdic (offset, len, result_str)   /* Get ebcdic string    */
int     offset;                 /* offset to string from current pif pointer*/
int     len;                    /* maximum number of source bytes           */
char    result_str[];           /* destination string                       */
```

Translate a printable EBCDIC zero-terminated string at the current offset in the packet into ASCII and move it to a C string. Unprintable characters are replaced with the character "`.`" The destination string ends with a "`\0`"even when the source doesn't. The return value is a pointer to the *end* (null) of the destination string. The PIF offset is not changed. If the menus force ASCII display, this calls `pif_get_ascii` instead.

```
char    *pif_get_lstring (offset, result_str)      /* Get Lstring           */
int     offset;                 /* Signed offset from current position      */
char    result_str[];           /* Return ASCIIz string here                */
```

Move an ASCII zero-terminated lstring from the current offset in the packet to a C string. An `lstring` starts with a length byte followed by that number of characters. Unprintable characters are replaced with the character "`.`". The destination string ends with a "`\0`"even when the source doesn't. The return value is a pointer to the *end* (null) of the destination string.

```
char    *pif_get_addr ()        /* return the data address                  */
```

Return the address of the field that is at the current PIF offset. This is a macro defined in `pi.h`.

```
char    pif_show_byte (prstr)
int     pif_show_word (prstr)
int     pif_show_word_hl( prstr)
long    pif_show_long (prstr)
long    pif_show_long_hl (prstr)
void    pif_show_2byte (prstr)
void    pif_show_4byte (prstr)
void    pif_show_6byte (prstr)
void    pif_show_nbytes_hex (prstr, n)
char    *prstr;                 /* A sprintf() control string               */
```

Create a new line in the **detail** view with the text given in `prstr` and the indicated data from the frame. The text should contain a formatting code like `%d` or `%x` indicating where the value should be printed. For the longword displays, the formatting code should be `%ld` or `%lx`. For `pif_show_2byte`, `pif_show_4byte`, and `pif_show_6byte` there should be 2, 4, and 6 formatting codes, respectively. For `pif_show_nbytes_hex`, there should be a single `%s` formatting code, and `n` specifies the number of bytes to display, from 1 to 99. The PIF offset is updated. As a convenience, the byte, word, and long routines return the value displayed as the function result.

```
void    pif_show_ascii (len, prstr) /* Show ASCII text                    */
int     len;                        /* Number of bytes to display         */
char    *prstr;                     /* control string with embedded %s    */
```

Create a new **detail** line from ASCII text starting at the current offset. The caller provides a `sprintf` control string whose embedded `%s` is replaced with the ASCII string copied from frame data using `pif_get_ascii`. The PIF offset is updated.

```
void    pif_show_ebcdic (len, prstr)  /* show ebcdic text                 */
int     len;                          /* number of bytes to display       */
char    *prstr;                       /* control string with embedded %s  */
```

Create a new **detail** line from EBCDIC text, starting at the current offset. The caller provides a `sprintf` control string whose embedded `%s` is replaced with the EBCDIC string copied from frame data using `pif_get_ebcdic`. (If ASCII translation is forced by the menus, this calls `pif_show_ascii` instead.) The PIF offset is updated.

```
void    pif_show_lstring (prstr)    /* show lstring
*/
char    *prstr;                     /*control string with an embedded
%s    */
```

Create a new **detail** line from an ASCII lstring, starting at the current offset. An `lstring` starts with a length byte followed by that number of characters. The caller provides a `sprintf` control string whose embedded `%s` is replaced with the string copied from frame data using `pif_get_lstring`. The PIF offset is updated.

```
void    pif_show_flag (prstr, mask)          /* show flag byte          */
char    *prstr;          /* title string: " = %d" is automatically added */
char    mask;            /* mask value indicating which bits to display  */
```

This routine displays the value of a byte with bit flags and sets up the correct information for subsequent calls to `show_flagbit`. The PIF offset is incremented by 1.

```
boolean pif_show_flagbit (bit, truestr, falsestr)     /* show flag bits  */
char    bit;                /* Bit mask for 1 or more bits               */
char    truestr[];          /* string to show if any masked bits are on  */
char    falsestr[];         /* string to show if all bits are off        */
```

This writes an 8-character field in the form "`....1...` <string>," indented as appropriate for the previous `pif_show_flag` call. If the `falsestr` is NULLP, the `truestr` is used for both cases. Return TRUE if any of the specified bits were on.

```
boolean pif_show_flagmask (maskbits, value, prstr) /* conditional show flags*/
char    maskbits;                                  /* Only check these bits    */
char    value;                                     /* Check for this value     */
char    *prstr;                                    /* Write this string if matched */
```

Write a **detail** line for a bit field only if the masked bits are a specified value. The line is written in the same format as for `pif_show_flagbit`. Return TRUE if the flag bits were the specified value and the line was written.

```
void  pif_show_date (prstr)                        /* show Unix-style date
*/
void  pif_show_date_hl (prstr)                     /* show Unix-style date
*/
char  *prstr;                              /* control string with embedded %s
*/
```

Create a new **detail** line from the text given in `prstr`, with the %s replaced with a readable date and time such as `"13-May-90 11:47:13"`. The date and time are taken from a 4-byte integer at the current PIF offset representing the number of seconds since 1/1/70 at midnight. The integer should be stored with the most significant byte first for `pif_show_date_hl` and with the least significant byte first for `pif_show_date`. The PIF offset is incremented by 4.

```
void    pif_show_space ()                  /* display a blank line          */
```

Write a blank line to the **detail** view.

```
void    pif_header (len, prstr)            /* Write a header line           */
int     len;                               /* Length of area to highlight   */
char    *prstr;                            /* Header string                 */
```

Output a header line to the **detail** view in the standard `"----- header_text -----"` format, followed by a blank line. Highlight data starting at the current offset for the length specified. This routine does not update the PIF offset. This routine saves the header string in the global called called `header_msg` so other routines can use it.

```
void   pif_trailer ()                          /* write a trailer line      */
```

Output a trailer line to the **detail** view that reports on how much of the frame data was used by the interpreter, based on the final position of the `PIF_offset`. This routine uses the header string saved by `pif_header`.

```
void  pif_autoscroll ()                        /* set autoscroll position   */
```

Make the next header line written to the **detail** view with `pif_header()` be the one that is scrolled to the top when **summary** highlighting is done. This is necessary only if the next header line is not the first for this registered PI.

```
char    *pif_line (len)        /* get detail line pointer                    */
int     len;                   /* Number of bytes to highlight and advance   */
```

Return a pointer to a **detail** line area. The caller also passes a length that causes the specified number of bytes (at the current PIF offset) to be highlighted in the hex view. The buffer pointer is advanced by the number of bytes specified. Beware of the side effects if you use this as the first argument to `sprintf();` no other arguments should depend on the buffer pointer because compilers differ in the order of argument evaluation!

```
void    pif_set (address)      /* set the PIF offset                         */
char    *address;
```

Set the PIF offset to the distance from the start of the frame data (`dlc_header`) to the specified address. This is a macro defined in `pi.h`.

```
void    pif_skip (delta)       /* move the PIF offset                        */
int     delta;
```

Add the signed delta to the current PIF offset. This is a macro defined in `pi.h`.

```
char    *pif_get_addr ()       /* return frame data address                  */
```

Return the address of the data item at the current PIF offset. This is a macro defined in `pi.h`.

Building a New Sniffer Analyzer

The programming environment for compiling and linking new PIs is that of Microsoft C Version 5.10 and Quick C, Version 2.00. No other C compilers are supported.

All the tools needed to write new protocol interpreters and to build new Sniffer analyzers are included on the hard disk, except for the Microsoft C compiler. To install the compiler, run the Microsoft compiler by inserting the "setup" disk in the drive and by typing

```
a:setup
```

You may accept the defaults for all the prompts except for two:

- Specify that you want the `LARGE` model library.
- Override the first pathname (for "bound executables") by specifying

```
c:\tools\mc\exe
```

All other files will then also go in directories within `c:\tools\mc`.

Once you have installed the compiler, the steps for writing your own PI and building a Sniffer are as follows:

- Edit the `initpi.c` file to register your PI.
- Create your source file (i.e., `mypi.c`).

- Copy `pisw<xxxx>.h` to `piswitch.h`, and edit the flags to choose PIs.
- Compile and link a new Sniffer analyzer by typing at the prompt

```
build mypi
```

To add the new Sniffer analyzer to the main Sniffer menu:

- Modify the HELP strings in the file `ensniffx.mnu` to give the correct help information for your new protocol interpreter.
- Copy `ensniff.exe` to `\ensniff\ensniffx.exe`.
- Copy `ensniffx.mnu` to `\config`.

Note that the *build* batch file tries to run the new Sniffer analyzer without adjusting the special memory maps. Large analyzers won't have enough memory to run. In that case, add the new analyzer to the main menu, and run it from there.

Files in the `newpi` sub-directory are listed in Figure 37.

pi.h	Standard PI symbols
piswitch.h	Flags for PI suite choices
network.h	Flags for the network type
pifdecl.h	Function type declarations
smb.h	SMB structure declaration
smbglobe.h	SMB header file
snmp.h	Symbols used in `netmgmt.c`
tcp.h	TCP header file
initpi.c	Protocol registration code
intnetbo.c	NetBIOS stub code
intsmbo.c	SMB stub code
netmgmt.c	Management information data base (MIB) for SNMP interpreter
tcpports.c	IP, TCP, UDP interpreter tables
xwinext.c	X Windows protocol interpreter extensions
sample.c	Source of the sample PI
***xx*SNIFFC.LIB**	Library of Sniffer analyzer core modules
***xx*SNIFFP.LIB**	Library of Sniffer analyzer PI modules
***xx*SNIFFX.MNU**	Menu items for selection menu
BUILD.BAT	Batch file to build the Sniffer analyzer
SAMPLE.*xx*C	Frame data for the sample PI
readme	Notes on building new sniffer protocol analyzers

*Figure 37. Files in the **newpi** directory.*

The batch file `build.bat` can be used to compile and link a PI that you write from scratch or a modified version of one of the "stub" interpreters:

```
@echo off

rem  This batch file ("build.bat") builds a new version of the Sniffer
rem  with user-written Protocol Interpreters.

rem  Copy pisw<xxxx>.h to piswitch.h and edit the flags to choose PIs.
rem  Edit the "initpi.c" file to register the new protocol interpreter.
rem  Supply the name of the new Protocol Interpreter source module as the
rem  argument to the batch file, as in "build myprot" or "build sample".

rem  If you are extending one of the stub interpreters for NETBIOS
rem  SMB, then you would say "build intnetbo" or "build intsmbo".
rem  If you are compiling more than one, you should modify this batch
rem  file appropriately, or perhaps use the Microsoft MAKE facility.

rem  We assume that Microsoft Version 5.1 compiler and tools
rem  are located in the path \tools\mc; see the "readme" file.

rem  If you get a linker error about /SE being invalid,
rem  you are using the wrong version of the linker.
rem  If you get a bogus compiler error like "cannot find pi.h",
rem  the config.sys file does not specify at least "files=15".

echo on
path c:\tools\mc\bin;c:\tools\mc\exe;c:\tools;c:\dos;c:\
set lib=c:\tools\mc\lib
set include=c:\tools\mc\include

cl /c /AL /Oat /J /Gs /Zp /Gt16 /G2 %1.c
if errorlevel 1 goto :end

cl /c /AL /Oat /J /Gs /Zp /Gt16 /G2 initpi.c
if errorlevel 1 goto :end

link initpi+%1,ensniff,ensniff,ensniffp+ensniffc /SE:500 /CP:1 /STACK:15000;
if errorlevel 1 goto :end

set enhelp=\ensniff
set ennames=\ensniff
ensniff
set enhelp=
set ennames=

:end
path=c:\tools;c:\dos;c:\
```

Figure 38. Batch program to build a new Sniffer executable.

In summary, the steps to take in writing a PI are:

- Modify `initpi.c` when writing a new PI (or when you wish to register your extensions to an existing PI).
- Write your C module, or modify the supplied stub routine.
- Run the `build` batch file.

An Example

Suppose that you want to write a new PI for the "sample" protocol that has simple fixed-format data, like this:

[1 byte]	Device number
[1 byte]	Command type
[2 bytes]	Number of segments
[20 bytes]	Name of owner (asciiz)

Suppose further that this protocol data is sent using 802.3 LLC and DSAP hex 44. Using the existing interpreter registrations as an example, the following lines would be added at the appropriate places in `initpi.c` to register the new demultiplexed interpreter:

```
/*  In the declaration section...                                          */

struct pi_data  *pi_data_sample;              /* ptr to our PI data       */
int              interp_sample (void *, int); /* our PI function          */

/* PICK ONE OF THE FOLLOWING AS APPROPRIATE FOR THE NETWORK                */
static int       sample_arcid = 0x44;         /* for ARCNET               */
static int       sample_sap = 0x44;           /* for TR, Ethernet, etc    */


/*  In the body of the initpi() function...                                */
/* PICK ONE OF THE FOLLOWING AS APPROPRIATE FOR THE NETWORK                */

pi_data_sample =
register_pi ("Sample", PITYPE_ARCID, 1, &sample_arcid, interp_sample, "SAMPLE: ");

pi_data_sample =
register_pi ("Sample", PITYPE_SAP,   1, &sample_sap,   interp_sample, "SAMPLE: ");
```

(Note that this code exists in `initpi.c` and can be actuated by changing the line `#define SAMPLE 0` to `#define SAMPLE 1`.)

In a new module, you would then write your PI with a structure something like the listing shown in Figure 39.

```
Protocol interpreter for the sample protocol

#define  USE_PIF   1            /* should we use the PIF routines? */

#include "pi.h"

extern struct pi_data *pi_data_sample;  /* our PI data */

struct  sample_header {                 /* the format of our frame data */
          char  device;
          char  command;
          int   nsegments;
          char  owner [20];
          };

int interp_sample (header, length)      /* our sample interpreter */

struct sample_header  *header;          /* pointer to our protocol header */
int                   length;           /* length of the remaining data */

{
    if (pi_data_sample->do_sum) {       /* summary line wanted? */
        sprintf (
            get_sum_line (pi_data_sample),      /* get a line buffer */
                "SMP Device = %d, Cmd = %02X",
                header->device, header->command);
        } /* end of summary line */

    if (pi_data_sample->do_int) {       /* detail lines wanted? */

#if USE_PIF                             /* show how to use PIF routines */

        /* Set up PIF globals */

        pif_init (pi_data_sample, header, length);

        /* Output the header line and highlight the whole header.
           Note that pif_header() does not alter the PIF offset. */

        pif_header (sizeof (struct sample_header),
            "Sample protocol data area");

        /* Display the fields.  Each routine advances the buffer pointer
           past the data item just displayed. */

        pif_show_byte ("Device number = %d");
        pif_show_byte ("Command type = %02X");
        pif_show_word ("Number of segments = %d");
        pif_show_ascii (20, "Owner = %s");

        /* Write out "End of. ." message & check for excess or missing data */

        pif_trailer ();

#else                                   /* Do detail without PIF routines */

        sprintf (
            get_int_line (pi_data_sample,           /* get a line buffer */
              (char *)&header->device - dlc_header,  /* highlight offset */
              2),                                    /* highlight length */
            "Device number = %d, command type = %02X",
            header->device, header->command);
```

Continued on next page...

Figure 39. Sketch for structure of a new protocol interpreter.

```
...continued from previous page.
        sprintf (
            get_int_line (pi_data_sample,              /* get a line buffer */
              (char *)&header->nsegments - dlc_header, /* highlight offset */
              2),                                      /* highlight length */
            "Number of segments = %d",
            header->nsegments);

        sprintf (
            get_int_line (pi_data_sample,              /* get a line buffer */
                (char *)header->owner - dlc_header,    /* highlight offset */
                20),                                   /* highlight length */
            "Owner = %20s",
            header->owner);

#endif

        } /* end of detail lines */

        /* If there were any embedded protocol after our header,
           we could call the interpreters here.  */

        return length;                  /* say that we used up all the data */
        }

/* end of sample.c */
```

Figure 39. Continued.

Generating lines for the **detail** views can be simplified by using the PIF formatting routines. That section of code, including the generation of header and trailer lines, might then look as shown in Figure 40.

```
    if (pi_data_sample->do_int) {          /* detail lines wanted            */

       /*  Set up PIF globals.                                                */

       pif_init(pi_data_sample, header, length);

       /*  Output the header line and highlight the whole header.
           Note that pif_header() does not alter the PIF offset.             */

       pif_header (sizeof (struct sample_header),
                   "Sample Protocol Data Area");

       /*  Display the fields; each routine advances the buffer pointer
           past the data item just displayed.                                */

       pif_show_byte ("Device number = %d");
       pif_show_byte ("Command type = %02X");
       pif_show_word ("Number of segments = %d");
       pif_show_ascii (20, "Owner = %s");

       /*  Write out "End of..." message and check for excess/missing data. */

       pif_trailer ();

       }                                   /* end of detail lines            */
```

*Figure 40. Use of PIF routines to format output for the **detail** view.*

You would then build and test the new Sniffer analyzer with the command `build sample`.

Augmenting Existing Protocol Interpreters

This section describes how to modify an existing PI as an alternative to writing one from scratch. There are several reasons why you may wish to augment or modify an available PI:

- There are local or proprietary changes to the protocol compared to the specification from which the PI was written.
- There are newly-added or proprietary extensions to the protocol. These are often in the form of additional "opcodes," "type codes," or "formats" added to an existing specification.
- To change the display format of the existing PI. The most common example is to change the selection or format of information displayed on the **summary line**.

In most cases, you can make the above changes to a PI and retain its useful parts, without having access to the source code. This presents a substantial advantage when modifying complex PIs that represent thousands of lines of code and many hours of testing.

The following are three basic techniques, you can use to modify a PI depending on its type:

- Some protocol interpreters call routines to process command codes or types they do not recognize. The C-language source for those routines, that normally only issues "unknown type" messages, is included with the Sniffer analyzer. You can modify those routines to process the new types and to allow the existing PI to process the original types.

  Currently, the NetBIOS, SMB, and X-Windows interpreters do this. They call routines in the following files to process unknown message types: `intnetbo.c`, `intsmbo.c`, and `xwinext.c`. Look at the source code supplied with the Sniffer analyzer for calling conventions.

- You can modify two tables to call your new PI. The file, `tcpports.c`, contains demultiplexing tables for protocols in the TCP/IP family and for adding IP, TCP, and UDP interpreters on well-known port numbers. The file, `netmgmt.c`, contains tables for *management information data bases* (MIBs) that can be interpreted by the Simple Network Management Protocol (SNMP) in the TCP/IP PI suite. You can find instructions for modifying these tables in comments in the code.

- You may write a PI that acts as a capture filter to the existing PI. Your PI will look at the protocol header to see if it handles that particular head type. If so, it interprets the data and returns. If not, it calls the existing PI.

  To modify PIs that are registered as type demultiplexers, simply change the registration in `initpi.c`. The `initpi.c` file will then call your routine instead of the original. Many other embedded PIs are called indirectly from a pointer variable and then from another PI. The pointer variable in the form

`piptr_xxxxxx` is initialized with the address of the PI in `initpc.c`. You can change the initialization to point to your capture filter routine.

If you wish to generate your own **summary line** instead of the one generated by the standard PI, and if there are no further embedded protocols, then you may use your routine for the **summary line** request and call the standard PI for the other cases. If there are further embedded PIs called by the standard PI, and you do not wish to duplicate the code that calls them, you can call the standard PI in all cases. If you are calling the standard PI, turn off the **summary line** request flag, as shown in the example below.

Another approach to generating your own **summary line** is to separately register your replacement PI. In this case, select the separate display flag you wish to see.

Example: Changing the TCP Summary Line

This example shows how to write a capture filter for the TCP interpreter (part of the TCP/IP PI) in order to change the formatting of the **summary line** while generating the lines for the **detail** view with the standard interpreter.

```
/* file: sample2.c */
/*****************************************************************************
SAMPLE2.C
This is an example capture filter to the TCP Protocol Interpreter that changes
the format of the summary line from that generated by the standard PI.

>>>  This is for Sniffer analyzer version 2.30, where interp_tcp() is
>>>  not called indirectly by interp_ip() using piptr_tcp.
>>>  Accordingly, we intercept at the IP level instead of the TCP level;
>>>  all occurrences of "interp_ip" are changed to "interp_ip2" in initpi.c.
*****************************************************************************/
#include "pi.h"

extern  struct pi_data  *pi_data_tcp;   /* ptr to PI data for TCP PI */
extern  int interp_ip (void *, int);    /* the original IP interpreter */

/*--- Our replacement IP interpreter. It gets control because all occurrences
   of "interp_ip" have been replaced by "interp_ip2" in initpi.c
---*/

int interp_ip2 (ip_ptr, length)         /* OUR REPLACEMENT IP INTERPRETER */
char    *ip_ptr;                        /* pointer to IP header */
int     length;                         /* length of remaining data */
{
int     tcp_sumflag, bytes_used, flags;
char    *ptr, *tcp_ptr;

    tcp_sumflag = pi_data_tcp->do_sum;          /* save TCP PI's summary flag
*/
    pi_data_tcp->do_sum = 0;                    /* temporarily turn it off */
    bytes_used = interp_ip (ip_ptr, length);    /* call IP then perhaps TCP */
    pi_data_tcp->do_sum = tcp_sumflag;          /* put back TCP's flag */
    if (tcp_sumflag                             /* if it was on, */
     && ip_ptr [9] == 6) {                      /*   and the embedded protocol
was TCP, */
        ptr = get_sum_line (pi_data_tcp);       /*   then generate our TCP
summary line. */
        tcp_ptr = ip_ptr +                      /* start of TCP header */
           ((ip_ptr [0] & 0xf) << 2);           /*   based on "IHL" IP field
*/
        ptr = stradd (ptr, "TCP ");             /* start with protocol name */
         /* Format the summary line here. As an example, we build a line
           that contains only the keywords for the TCP flags that are set.
           If we were serious, we'd define structures for the headers and
           probably format other fields as well. */
        flags = tcp_ptr [13];                   /* the TCP flags byte. */
        if (flags & 0x01)
            ptr = stradd (ptr, "FIN ");         /* data finished flag */
        if (flags & 0x02)
            ptr = stradd (ptr, "SYN ");         /* synchronize flag */
        if (flags & 0x04)
            ptr = stradd (ptr, "RST ");         /* reset flag */
        if (flags & 0x08)
            ptr = stradd (ptr, "PSH ");         /* push flag */
        if (flags & 0x10)
            ptr = stradd (ptr, "ACK ");         /* Acknowledge flag */
        if (flags & 0x20)
            ptr = stradd (ptr, "URG ");         /* urgent flag */
        }
    return bytes_used;
    }
/* end of sample2.c */
```

Programming and Debugging Hints

- The `LARGE` memory model is used throughout the Sniffer analyzer and must be used by your PIs. This means that pointers are 4 bytes and integers are 2 bytes, so that the symbol `NULLP`, not `0`, should always be used to represent a null pointer.

- Extreme care should be taken in writing PIs, especially in the manipulation of pointers used as targets. There is no hardware memory protection, and with 4-byte pointers, there is no segment addressability limitation; every byte in the machine is vulnerable to a wayward pointer. Be particularly careful that incorrect or even totally random protocol data will not cause your interpreter to overflow strings, to access invalid memory areas, or to loop.

- The Microsoft Codeview debugger is unlikely to be a useful tool because of the size of the Sniffer analyzer executable module, but `SYMDEB` works just fine.

- Avoid the use of `printf()` for debugging messages since the cursor is off the screen most of the time and the messages will not be seen. You can instead insert messages into the debugging window using the similarly-called `debug_msg()` function; use **Shift-F1** to pop up the debugging window. You can also insert debugging messages into the **detail** view by using `get_int_line()`. If you must use `printf()`, specify `DEBUG` as a command line parameter when invoking the Sniffer analyzer and the cursor will be left somewhere on the screen, but the output will destroy the screen formatting.

- If you are decoding headers or data that are large, beware of interpreting invalid information beyond the end of the stored data. If the data were captured in "partial frame" mode, the data present may be less than the entire frame; the global `bytes_not_stored` indicates how much data is not present.

- Your PI should not access data beyond the end of what has been stored for the frame. (Remember that some of the frame's data may have been discarded if the partial-frame size was other than "whole frame" when the data was collected.) If you are using a PIF routine and it detects that it is about to access locations beyond the end of the frame data, it puts a **frame too short** message into the **detail** view and it does not return to your PI.

- Be careful not to store more than `MAX_SUM_LINE` characters in a **summary** line buffer or more than `MAX_INT_LINE` characters in a **detail** line buffer, regardless of what the frame data might be.

- Remember that the 80286 or 80386 processor that is used in the Sniffer analyzer stores integers in low-high format. Depending on your protocol, you may have to reverse integers for printing or calculation. The PIF routines ending in `_hl` are useful in that case.

- The Microsoft Version 5.10 compiler supports argument type checking using ANSI standard function prototyping. The `#include` files provided with the Sniffer analyzer have declarations for all the documented functions, and you are encouraged to add declarations of your new functions.

- The `/J` flag has been used to compile all modules that make the default for character variables `unsigned`.
- As discussed in the Microsoft C *User's Guide*, the `config.sys` file in the root directory of the hard disk must specify at least `FILES=15`. If not, misleading errors like "`Cannot find xxx.h`" will result.
- If you are using the PIF routines, it is necessary to use `pif_save()` and `pif_restore()` only if you are calling embedded PIs and you wish to generate more **detail** view lines from the current interpreter after the embedded interpreter has returned.

Appendixes

Network General

Appendixes

Appendix A. Glossary

1BASE5 The implementation of the IEEE 802.3 (StarLAN) standard using 1 megabit per second transmission on a baseband medium whose maximum segment length is 500 meters.

10BASE2 The implementation of the IEEE 802.3 (Ethernet) standard using 10 megabit per second transmission on a baseband medium whose maximum segment length is 185 meters.

10BASE5 The implementation of the IEEE 802.3 (Ethernet) standard using 10 megabit per second transmission on a baseband medium whose maximum segment length is 500 meters.

10BASE-T The implementation of the IEEE 802.3 (Ethernet) standard using 10 megabit per second transmission on a baseband medium. The standard provides a means for attaching AUI-compatible devices to 24 gauge, unshielded twisted pair cable, instead of the usual coaxial media.

3COM 3+ A networking system from 3COM Corporation using parts of the XNS and Microsoft/IBM PC LAN program protocols.

3PLUS 3COM's implementation of XNS and interpreted by the XNS PI suite.

802.2 The IEEE standards designation for the LLC sublayer protocol that provides both datagram and reliable connection transmission.

802.3 The IEEE standards designation for the CSMA/CD network access method. Similar to (and often used interchangeably with) Ethernet.

802.5 The IEEE standards designation for the token ring network access method.

AARP AppleTalk Address Resolution Protocol. For outgoing packets, supplies the hardware destination address corresponding to a higher-level protocol address, and filters incoming packets to pass only those that are broadcast or specifically addressed to it. Interpreted in the AppleTalk PI suite.

AC Access control. A DLC byte on IEEE 802.5 token ring networks that contains the token indicator and frame priority information.

ACSE Association Control Service Element. An ISO application-level protocol interpreted in the ISO PI suite.

ACTPU Activate Physical Unit. An SNA message sent to start a session.

ADSP AppleTalk Data Stream Protocol. A connection-oriented protocol providing a reliable, full-duplex, byte-stream service between any two sockets on an AppleTalk internet, ensuring in-sequence, duplicate-free delivery of data over its connections. Interpreted in the AppleTalk PI suite.

AEP AppleTalk Echo Protocol. See Echo.

AFP AppleTalk Filing Protocol. A presentation-level protocol for access to remote files. Interpreted in the AppleTalk PI suite.

ALAP	AppleTalk Link Access Protocol. See LAP.
API	Application Program Interface. The specification of functions and data used by one program module to access another; the programming interface that corresponds to the boundary between protocol layers.
APPC	Advanced Program-to-Program Communications. A communications system used to communicate between transaction programs on IBM computers; APPC uses the LU 6.2 subset of SNA.
ARCNET	A baseband token-passing network originally designed by the Datapoint Corporation that communicates among up to 255 stations at 2.5 Mbps.
ARP	Address Resolution Protocol. (1) Interpreted in the Banyan VINES. PI suite (2) A protocol within TCP/IP for finding a node's DLC addresses from its IP address. Interpreted in the TCP/IP PI suite.
ASCII	American Standard Code for Information Interchange. A mapping between numeric codes and graphical characters used almost universally for all personal computer and non-IBM mainframe applications.
ASN.1	Abstract Syntax Notation One. A set of conventions governing the ISO presentation layer. Interpreted in the ISO PI suite.
ASP	AppleTalk Session Protocol. A general protocol, built upon ATP, providing session establishment, maintenance, and tear-down, along with request sequencing. Interpreted in the AppleTalk PI suite.
ATP	AppleTalk Transaction Protocol. Provides a loss-free transaction service between sockets, allowing exchanges between two socket clients in which one client requests the other to perform a particular task and report the result. Interpreted in the AppleTalk PI suite.
AUI	Attachment Unit Interface. Drop cable for Ethernet.
Background Service	A protocol transmitted by a Matchmaker frame in Banyan VINES.
Baseband	A modulation technique that sends data bits without using a much higher carrier frequency. The entire bandwidth of the transmission medium is used by one signal.
BIND	An SNA message sent to activate a session between LUs.
BIS	Bracket Initiation Stopped. An SNA message sent to indicate that the sending station will not attempt to initiate any more brackets.
BNC	A standardized coaxial cable connector; used for Thin Ethernet ("Cheapernet") cables and ARCNET networks.
BOOTP	Boot Protocol. A protocol within TCP/IP that is used for downloading initial programs into networked stations and interpreted in the TCP/IP PI suite.

Broadband A modulation technique that sends data bits encoded within a much higher radio-frequency carrier signal. The transmission medium may be shared by many simultaneous signals since each one only uses part of the available bandwidth.

Broadcast (1) A message directed to all stations on a network or collection of networks. (2) A destination address that designates all stations.

CCITT International Consultative Committee for Telephony and Telegraphy. CCITT is a member of the International Telecommunications Union (ITU) that is, in turn, a specialized body within the United Nations. It sponsors a number of standards dealing with data communications networks, telephone switching standards, digital systems, and terminals.

CGA Color Graphics Adapter. The interface between a personal computer and a medium-resolution color monitor.

CLNS Connectionless Network Service Protocol (also called ISO IP). Interpreted in the ISO PI suite.

CMIP Command Management Information and Services Protocol. When used with TCP/IP, it is also known as CMOT.

CMOT Common Management Information and Services Protocol Over TCP. A management protocol for networks; it uses ASN.1 encoding. Interpreted in the TCP/IP and ISO PIs.

Courier A presentation-level protocol in XNS (similar to RPC in the Sun protocol family); it delivers data to such application-level protocols as XNS Printing, XNS Filing, or XNS Clearinghouse.

CRC Cyclic Redundancy Check. A check-word, typically two or four bytes at the end of a frame, used to detect errors in the data portion of the frame.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. The algorithm used in LocalTalk networks to control transmission.

CSMA/CD Carrier Sense Multiple Access with Collision Detection. The algorithm used by IEEE 802.3 and Ethernet networks to control transmission.

CTERM Command Terminal. A protocol within DECnet for communicating with generic intelligent terminals, that is, a virtual terminal protocol. Interpreted in the DECnet PI suite.

DAP Data Access Protocol. The DECnet protocol that provides remote file access; interpreted in the DECnet PI suite.

DB-9 A 9-pin standardized connector used in personal computers for a token ring network connection (female), serial I/O port (male), and RGBI output. Also used for LocalTalk.

DB-15 A 15-pin standardized connector used to connect to the drop cable of an IEEE 802.3 or Ethernet transceiver.

DB-25 A 25-pin standardized connector used in personal computers for parallel output ports (female connector on IBM PC chassis) or for serial I/O ports (male connector on IBM PC chassis).

DCE Data Circuit-terminating Equipment (also called Data Communications Equipment). On a serial communications link, the device that connects the DTEs into the communication line or channel.

DDP Datagram Delivery Protocol. Extends the services of the underlying LAP protocol to include an internet of interconnected AppleTalk networks, with provision to address packets to sockets within a node. Interpreted in the AppleTalk PI suite.

DFC Data Flow Control. An SNA subprocess for reliable message transfer.

DIS Draft International Standard.

DISC Disconnect. An LLC non-data frame indicating that the connection established by an earlier SABM or SABME is to be broken.

DIX DEC/Intel/Xerox. Used to refer to an early version of Ethernet.

DLC Data Link Control. The lowest protocol level within the transmitted network frame; fields typically include the Destination address, and Source address, and perhaps other control information.

DLL Downline load. A protocol within the Datapoint RMS family used for downloading initial programs into networked stations.

DM Disconnected Mode;. An LLC message acknowledging that a previously established connection has been broken.

DNS Domain Name Service. A protocol within TCP/IP for finding out information about resources using a database distributed among different name servers. Interpreted in the TCP/IP PI suite.

DOS Disk Operating System. The most common operating system for IBM-compatible personal computers.

DRP DECnet Routing Protocol. The lowest-level DECnet protocol, concerning with moving packets from endnodes through routers to other endnodes. ("Routing" in DNA terminology corresponds to the ISO model's "Network" layer).

DSAP Destination Service Access Point. The LLC SAP for the protocol expected to be used by the destination station in decoding the frame data.

DTE Data Terminal Equipment. On a serial communications link, a generic term used to describe the end-user machine.

EBCDIC Extended Binary-Coded-Decimal Interchange Code. A mapping between numeric codes and graphical characters used for IBM mainframe computers and communications protocols defined by IBM.

Echo (1) A protocol within XNS used to verify the existence of a host, and for a response indicating routing to it; interpreted in the XNS PI suite.
(2) A simple protocol within AppleTalk that allows any node to send a datagram to any other node and receive an echoed copy of that packet in return, in order to verify the existence of that node or to make round trip delay measurements. Interpreted in the AppleTalk PI suite.
(3) A protocol transmitted by a Matchmaker frame in Banyan VINES.

EGP Exterior gateway protocol. A protocol within TCP/IP used to exchange routing information among gateways belonging to the same or different systems. A generalization of GGP.

ELAP See LAP.

Error A protocol within XNS by which a station reports that it has received (and is discarding) a defective packet; interpreted in the XNS PI suite.

ES-IS Routing End-System to Intermediate-System Routing. A protocol within the ISO family used to exchange routing information between gateways and hosts. Interpreted in the ISO PI suite.

Ethernet A CSMA/CD network standard originally developed by Xerox; similar to (and often used interchangeably with) the IEEE 802.3 standard.

Ethertype A 2-byte protocol-type code in Ethernet frames used by several manufacturers but independent of the IEEE 802.3 standard.

FC Frame control. On a token ring network, the DLC byte that contains the frame's type.

FCS Frame check sequence. A redundant check field used to increase the probability of error-free transmission on the network.

FID Format Identification. A field in the SNA Transmission header indicating the type of nodes participating in the conversation. LU 6.2 nodes are type 2.

FMD Function Management Data. A class of data embedded at the start of SNA RUs.

FMH Function Management Header. The header part of SNA FMD containing addressing and transmission control information.

FOUND Foundation Services. A protocol within DECnet used for primitive terminal-handling services; interpreted in the DECnet PI suite.

Frame The multi-byte unit of data transmitted at one time by a station on the network; synonymous with Packet.

FRMR Frame Reject. An LLC command or response indicating that a previous frame had a bad format and is being rejected. The REJ frame contains five bytes of data explaining why and how the previous frame was bad.

FRP Fragmentation Protocol. Breaks up and reassembles network-layer packets so that they are acceptable to the data-link protocol and the underlying physical medium; used on networks whose physical medium is ARCNET. interpreted in the Nestar PLAN Series and the Banyan VINES PI suites.

FS Frame status. A byte appended to a token ring network frame following the CRC. It contains the Address Recognized and Frame Copied bits.

FS CMD Fileserver commands. A protocol used by Nestar to issue commands from client stations to servers.

FTAM File Transfer, Access and Management. An application-level protocol within the ISO suite, on top of ACSE.

FTP File Transfer Protocol.
(1) A protocol transmitted by a Matchmaker frame in Banyan VINES.
(2) A protocol based on TCP/IP and Telnet for reliable file transfer. Interpreted in the TCP/IP PI suite.

Functional address A limited broadcast destination address for IEEE 802.5 token ring networks. Individual bits in the address specify attributes which station eligible to receive the frame should have. Similar to "multicast address."

GGP Gateway-to-gateway protocol. A protocol within TCP/IP used to exchange routing information between IP gateways and hosts; interpreted in the TCP/IP PI suite. See also EGP.

Hub A concentrator and repeater for the StarLAN or the ARCNET network. For StarLAN, it is more properly known as a Network Hub Unit or as a Network Extension Unit.

I Information. An LLC, HDLC, or SDLC frame type used to send sequenced data that must be acknowledged.

ICMP Internet Control Message Protocol. A protocol within TCP/IP used principally to report errors in datagram transmission. Interpreted in the, TCP/IP PI suite.

ICP Internet Control Protocol. Used to broadcast notification of errors and to note changes in network topology in Banyan VINES. Interpreted in XNS PI suite.

IDP Internet Datagram Protocol. Delivers to an internet address a single frame as an independent entity, without regard to other packets or to the addressee's response.

IEEE Institute of Electrical and Electronics Engineers, Inc. Standards documents are available from them at 345 East 47th Street, New York, NY 10017.

IOB Input/Output block protocol. Used by Nestar to make virtual disk requests from client station to servers.

IONET Input/Output Network. A device message protocol used by Datapoint.

IP	Internet Protocol. The lowest-level protocol under TCP/IP that is responsible for end-to-end forwarding and long packet fragmentation control. Interpreted in the TCP/IP PI suite. A similar protocol is interpreted in the Banyan VINES PI. See also IPX and ISO PI suites.
IPC	Interprocess Communication Protocol. A transport-level protocol in Banyan VINES, providing reliable message service and unreliable datagram service; interpreted in the Banyan VINES PI suite.
IPX	Internet Protocol. Novell's implementation of Xerox Internet Datagram Protocol; interpreted in the Novell NetWare PI suite.
ISO	International Organization for Standardization. (1) a consortium that is establishing a suite of networking protocols; (2) the protocols standardized by that group.
ISODE	ISO Development Environment. Protocol for transmitting higher-level ISO protocols over a network whose lower levels are handled by TCP/IP. Interpreted in the TCP/IP and ISO PI suites.
ISO IP	The ISO standard Internet Protocol. Interpreted in the ISO PI suite.
KSP	Kiewit Stream Protocol. A transport protocol resembling TCP developed at Dartmouth College for the support of terminal emulators connected to AppleTalk networks; interpreted in the AppleTalk PI suite.
LAN	Local Area Network. The hardware and software used to connect computers together in a limited geographical area.
LAP	Link Access Protocol. The logical level protocol for AppleTalk. It exists in two variants: ELAP (for Ethernet) and LLAP (for LocalTalk networks). Interpreted in the AppleTalk PI.
LAPB	Link Access Procedure, Balanced. A subset of HDLC.
LLAP	See LAP.
LAT	Local Area Transport. The DECnet protocol that handles multiplexed terminal (keyboard and screen) traffic to and from timesharing hosts. Interpreted in the DECnet PI suite.
LLC	Logical Link Control. A protocol that provides connection control and multiplexing to subsequent embedded protocols; standardized as IEEE 802.2 and ISO/DIS 8802/2.
LOOP	Loopback protocol. A protocol under Ethernet for sending diagnostic probe messages.
LSA	Lost Subarea. An SNA error condition.
LU 6.2	Logical Unit 6.2. A subset of the SNA protocols used for peer-to-peer communications between computers.
LUSTAT	Logical Unit Status. An SNA message used to send status information.

MAC Medium Access Control. The protocol level that describes network management frames sent on the 802.5 token ring. Most MAC frames are handled transparently by the network adapter.

Mail Service Protocol used (in conjunction with StreetTalk) for the transmission of messages in the VINES distributed electronic mail system. Interpreted in the Banyan VINES PI suite.

Manchester encoding A data encoding technique that uses a transition at the middle of each bit period that serves as a clock and also as data.

MAP Manufacturing Automation Protocol. An emerging multi-layer networking protocol developed primarily by General Motors for manufacturing control applications.

Matchmaker Protocol used by the VINES service that provides high-level program-to-program communication, including translation as necessary to match the conventions of sender's and receiver's formats. Matchmaker is descended from XNS Courier. Interpreted in the Banyan VINES PI suite.

MAU Multiple Access Unit (also Medium Attachment Unit). The wiring concentrator or transceiver used for attaching stations connected to the network.

MIB Management Information Data Base. The structured database of network statistical information used by the SNMP and CMIP protocols.

MOP Maintenance Operations Protocol. A protocol under DECnet for remote testing and problem diagnosis, interpreted in the DECnet PI suite.

MOUNT A protocol developed by Sun Microsystems for handling request access checking and user validation. It is used in conjunction with NFS. Interpreted in the Sun PI suite.

Multicast (1) A message directed to a subset of the stations on a network or collection of networks.
(2) A destination address that designates such a subset.

N(R) Receive sequence number. An LLC or HDLC field for I frames that indicates the sequence number of the next frame expected; all frames before N(R) are thus implicitly acknowledged.

N(S) Send sequence number. An LLC or HDLC field for I frames that indicates the sequence number of the current frame within the connection.

NBP (1) Name-Binding Protocol. Used in AppleTalk networks to permit network users to use character names for network services and sockets. NBP translates a character-string name within a zone into the corresponding socket address. Interpreted in the AppleTalk PI suite.
(2) NetBios Protocol. Used in 3Com 3+ Open software. Interpreted in the XNS PI suite.

NC Network Control. An SNA subprocess.

NCP NetWare Core Protocol. Novell's application-level protocol for the exchange of commands and data between file servers and workstations. Interpreted in the Novell NetWare PI suite.

ND Network Disk. A protocol within the Sun NFS family used to access virtual disks located remotely across the network. Interpreted in the TCP/IP PI suite.

NetBIOS Network Basic I/O System.
(1) A protocol implemented by the PC LAN Program to support symbolically named stations and the exchange of arbitrary data.
(2) The programming interface (API) used to send and receive NetBIOS messages.
There exist several different and incompatible implementations of NetBIOS, and separate PIs for them, including the IBM and the TCP/IP PI suites.

NETBLT Network Block Transfer. A protocol within earlier version of TCP/IP (but not interpreted in the TCP/IP PI suite).

NetWare The networking system designed by Novell Inc. and the protocols used therein.

Network Management A protocol transmitted by a Matchmaker frame in Banyan VINES.

NEU Network Extension Unit. A concentrator and repeater for StarLAN networks.

NFS Network File System. A protocol developed by Sun Microsystems for requests and responses to a networked file server; interpreted in the Sun PI suite.

NHU Network Hub Unit. A concentrator and repeater for StarLAN networks.

NICE Network Information and Control Exchange. The DECnet protocol for network management; interpreted in the DECnet PI suite.

NSP Network Services Protocol. The DECnet protocol that provides reliable message transmission over virtual circuits interpreted in the DECnet PI suite.

OpenNET A networking system from the Intel Corporation that parts of the OSI standards and components of the Microsoft/IBM PC LAN program, interpreted in the ISO PI suite.

OSI Open Systems Interconnection. A generalized model of a layered architecture for the interconnection of systems.

Packet The multi-byte unit of data transmitted at one time by a station on the network; synonymous with Frame.

PAP Printer Access Protocol. A protocol within AppleTalk that uses ATP XO commands to create a stream-like service for communication between user stations and the Apple LaserWriter or similar stream-based devices. Interpreted in the AppleTalk PI suite.

PC*I	Personal Computer Integration. Data General's nomenclature for their networking system. Protocols used include the ISO IP and TP4 levels and the Microsoft/IBM PC LAN program SMB protocols; interpreted in the ISO PI suite.
PCF	Physical Control Fields. The part of the token ring DLC header that includes the AC and FC fields.
PDU	Protocol Data Unit. The data delivered as a single unit between peer processes on different computers.
PEP	Packet Exchange Protocol. A protocol within the XNS family used to exchange datagrams. Interpreted in the XNS/MS-Net PI suite.
PI	Protocol Interpreter. A program that knows the frame format and transaction rules of a communications protocol and can decode and display frame data.
PMAP	Port Mapper. A protocol developed by Sun Microsystems for mapping RPC program numbers to TCP/IP port numbers; interpreted in the Sun PI suite.
PUP	PARC Universal Packet. A type of Ethernet packet formerly used at the Xerox Corporation's Palo Alto Research Center. Interpreted in the XNS/MS-Net and the TCP/IP PIs but not included in their protocol diagrams since no longer in regular use.
RARP	Reverse Address Resolution Protocol. A protocol within TCP/IP for finding a node's IP address given its DLC address. Interpreted in the TCP/IP PI suite.
RDP	Reliable datagram protocol. A protocol within earlier version of TCP/IP (but not interpreted in the TCP/IP PI suite).
REJ	Reject. An LLC frame type that requests retransmission of previously sent frames.
REM	Ring Error Monitor. A station on the 802.5 token ring network that collects MAC-level error messages from the other stations.
RFC	Request For Comment. Designation used in DoD/TCP protocol research and development.
RG-58	The designation for 50-ohm coaxial cables used by Cheapernet (thin Ethernet).
RG-59	The designation for 75-ohm coaxial cables used by PC Network
RG-62	The designation for 93-ohm coaxial cables used by ARCNET.
RGBI	Red-Green-Blue-Intensity. An interface used for attaching a color monitor to a personal computer; DB-9 connectors are typically used.
RH	Request/response header. An SNA control field prior to a Request Unit or Response unit.

RI Routing Information. A protocol at the logical link level for devices operating on the token ring. Interpreted by the token ring, Ethernet, StarLAN, and PC Network Sniffers independent of other PIs.

RII Routing Information Indicator. If the first bit in the source address field of a token ring frame is 1, then the data field begins with Routing Information. Interpreted by the token ring, Ethernet, StarLAN, and PC Network Sniffers independent of other PIs.

RIP Routing Information protocol. A protocol within the XNS and TCP/IP families used to exchange routing information among gateways. Interpreted in the XNSt PI suite and in the TCP/IP PI suite.

RJ-45 The designation for the 8-wire modular connectors used for StarLAN networks. It is similar to, but wider than, the standard phone modular connectors.

RMS Resource Management System. A set of protocols used by Datapoint to communicate from client stations to servers.

RNR Receive Not Ready. An LLC and HDLC command or response indicating that transmission is blocked.

Router (1) A protocol transmitted by a Matchmaker frame in Banyan VINES.
(2) An internet linking device operating at network layer 3.

RPC Remote Procedure Call. A protocol for activating functions on a remote station and retrieving the result. Interpreted in the Sun PI suite. A similar protocol exists in Xerox XNS.

RPL Remote Program Load. A protocol used by IBM on the IEEE 802.5 token ring network to download initial programs into networked stations. Interpreted in the IBM PI suite.

RPS Ring Parameter Server. A station on a token ring network that maintains MAC-level information about the LAN configuration such as ring numbers and physical location identifiers.

RR Receive ready. An LLC non-data frame indicating readiness to receive data from the other station.

RS-232C Recommended Standard 232. EIA standard defining electrical characteristics of the signals in the cables that connect a DTE and a DCE.

RSTAT Remote status. A protocol with the Sun NFS family used to exchange statistics on network activity; interpreted in the Sun PI suite.

RTMP Routing Maintenance Protocol. Used in AppleTalk networks to allow bridges or internet routers dynamically to discover routes to the various networks of an internet. A node that is not a bridge uses a subset of RTMP (the RTMP stub) to determine the number of the network to which it is connected and the node IDs of bridges on its network. Interpreted in AP-1310 AppleTalk.

RTP Routing Update Protocol. Used to distributed network topology information. Interpreted in the Banyan VINES PI suite.

RU Request Unit/Response unit. The part of an SNA frame after the RH that contains the details of a request or its response.

RUnix Remote Unix. A protocol under TCP/IP for issuing remote requests over the network to a UNIX host.

SABM Set Asynchronous Balanced Mode. An LLC non-data frame requesting the establishment of a connection over which numbered I frames may be sent.

SABME Set Asynchronous Balanced Mode (Extended). SABM with two more bytes in control field. Used in LAPB.

SAP Service Access Point.
(1) A small number used by convention or established by a standards group, that defines the format of subsequent LLC data; a means of demultiplexing alternative protocols supported by LLC.
(2) Service Advertising Protocol; a protocol in Novell NetWare by which a server makes itself known to potential network users. Interpreted in the Novell NetWare PI suite.

SBI Stop Bracket Initiation. An SNA message sent to request that the other station not initiate any more brackets.

SC Session Control. An SNA subprocess for establishing and maintaining connections.

SCP Session Control Protocol. The DECnet protocol concerned with the establishment of virtual circuits over which NSP transfers data; interpreted in the DECnet PI suite.

SDLC Synchronous Data Link Control. An older serial communications protocol that was the model for LLC and with which it shares many features.

SESSION Name for the session-level protocol in the ISO series, interpreted in the ISO PI suite.

Sever A protocol transmitted by a Matchmaker frame in Banyan VINES.

SIG Signal. A high-priority SNA message used to request permission to send.

SMB Server Message Block. A message type used by the IBM PC LAN Program to make requests from a user station to a server and receive replies. Many of the functions are similar to those made by an application program to DOS or to OS/2 running on a single computer. SMB is part of the protocol family that for DOS machines is called MS-NET and for OS/2 machines is called The LAN Manager. Under the IBM PC LAN Program, SMBs are sent as data within NetBIOS frames, but in other context may be transported differently. The OS/2 version of SMB contains extensions not present in the DOS version. Both versions are interpreted in the IBM , XNS, TCP/IP, ISO, DECnet, Nestar, and Banyan Vines PI suites.

SMTP Simple Mail Transfer Protocol. A protocol within TCP/IP for reliable exchange of electronic mail messages. Interpreted in the TCP/IP PI suite.

SNA Systems Network Architecture. A complex set of protocols used by IBM for network communications, particularly with mainframe computers. Interpreted in the IBM PI suite.

SNAP Sub-Network Access Protocol (also sometimes Sub-Network Access Convergence Protocol). Interpreted in the TCP/IP PI suite and the AppleTalk PI suite.

SNMP Simple Network Management Protocol. Interpreted in the TCP/IP PI suite .

SNRM Set Normal Response Mode. Place a secondary station in a mode that precludes it from sending unsolicited frames. The primary station controls all message flow. Used in SDLC.

SNRME Set Normal Response Mode (Extended). SNRM with two more bytes in control field. Used in SDLC.

Socket A logically addressable entity or service within a node, serving as a more precise identification of sender or recipient.

SPP Sequenced Packet Protocol. A virtual-circuit connection-oriented protocol in XNS.

SPP Sequenced Packet Protocol.
(1) The subset of XNS that supports reliable connections using sequenced data; interpreted in the XNS PI suite. A variant called SPX is used in Novell NetWare.
(2) Sequenced Packet Protocol. The transport-level protocol to provide virtual connection service in Banyan VINES, based upon the protocol of the same name in XNS; interpreted in the Banyan VINES PI suite.

SPX Sequential Packet Exchange. Novell's version of the Xerox protocol called SPP; interpreted in Novell NetWare PI suite.

SQE Signal Quality Error. The 802.3/Ethernet collision signal from the transceiver.

SQE TEST The SQE signal generated by the transceiver at the end of a transmitted frame to check the SQE circuitry. Also known as *Heart Beat* in Ethernet.

SSAP Source Service Access Point. The LLC SAP for the protocol used by the originating station.

SSCP System Services Control Point. An SNA identification of communications management functions.

StarLAN A network developed by AT&T Bell Labs and based upon a derivative of the CSMA/CD (Ethernet) network standard originally developed by Xerox; similar to (and often used interchangeably with) the IEEE 802.3 standard.

StreetTalk Protocol used in Banyan VINES to maintain a distributed directory of the names of network resources. In VINES names are global across the internet and independent of the network topology. Interpreted in the Banyan VINES PI suite.

SUA Stored Upstream Address. The network address of a token ring station's nearest upstream neighbor. Texas Instruments calls this the UNA.

S Supervisory. An LLC, HDLC, or SDLC frame type used for control functions.

Talk A protocol transmitted by a Matchmaker frame in Banyan VINES.

TC Transmission Control. An SNA subprocess.

TCP Transmission Control Protocol. The connection-oriented byte-stream protocol within TCP/IP that provides reliable end-to-end communication by using sequenced data sent by IP. Interpreted in the TCP/IP P.

TCP/IP Transmission Control Protocol/Internet Protocol. A suite of networking protocols developed originally by the US Government for Arpanet and now used by several LAN manufacturers. (The individual TCP/IP protocols are listed separately in this Glossary.).

Telnet Protocol for transmitting character-oriented terminal (keyboard and screen) data. Interpreted in the TCP/IP PI suite.

TFTP Trivial File Transfer Protocol. A protocol within TCP/IP used to exchange files between networked stations. Interpreted in the TCP/IP PI suite.

TH Transmission header. The initial part of an SNA frame immediately following the LLC header.

Token A small message used in some networks to represent the permission to transmit; it is passed from station to station in a predefined sequence.

Token bus A type of LAN where all stations can hear what any station transmits and where permission to transmit is represented by a token sent from station to station.

Token ring A type of LAN where stations are wired in a ring and each can directly hear transmissions only from its immediate neighbor. Permission to transmit is granted by a token that circulates around the ring.

TP Transport-level Protocol. It exists in alternate forms, depending on the services it assumes are provided to it by the network level below it. TP 0 assumes the the connection is maintained at the lower level, while TP 4 assumes a connectionless network protocol, so that functionality for the establishment and maintenance of a connection are included in the transport protocol. Levels 0, 2, and 4 are interpreted in the ISO PI suite.

TRLR Trailer format. Variant of IP in which the protocol headers follow rather than precede the user data.

TS Transmission Services. An SNA subprocess.

UA	Unnumbered Acknowledgment. An LLC frame that acknowledges a previous SABME or DISC request.
UB	Interpreted in the XNS PI suite.
UDP	User Datagram Protocol. A protocol within TCP/IP for sending unsequenced data frames not otherwise interpreted by TCP/IP.
UI	Unnumbered Information. An LLC, HDLC, or SDLC frame type used to send data without sequence numbers.
UNA	Upstream Neighbor Address. The network address of a token ring station's nearest upstream neighbor. IBM calls this the SUA.
UNIX	A popular portable operating system written (and trademarked) by AT&T.
VINES	VIrtual NEtwork Software. The networking operating system developed by Banyan Systems Inc. and the protocols used therein. Notable components are StreetTalk and MatchMaker.
VMTP	Versatile Message Transaction Protocol (proposed).
VTP	Virtual Terminal Protocol.
V.35	A CCITT wideband interface recommendation.
WAN	Wide Area Network. A network that uses common carrier-provided lines.
X.25	A CCITT recommendation that defines the standard communications protocol for access to packet-switched networks.
X.400	ISO standard protocol for electronic mail. Interpreted in the ISO PI suite.
XID	Exchange Identification. An LLC unnumbered frame type used to negotiate what LLC services will be used during a connection.
XNS	Xerox Network Systems. A family of protocols standardized by Xerox; in particular the Internet Transport Protocols. Documents are available from Xerox Document Systems Business Unit, 475 Oakmead Parkway, Sunnyvale, CA 94086. XNS is interpreted in the Novell NetWare, the XNS/MS-Net, and the Nestar PLAN Series PIs).
XWIN	Protocol for the management of high-resolution color windows at workstations, originated by MIT, DEC and IBM and subsequently transferred to a consortium of vendors and developers. Version 11 is interpreted in the X Windows PI suite, including some extensions added by individual vendors, for example DEC's DECWindows.
YP	Yellow Pages. A protocol developed by Sun Microsystems for implementing a distributed resource look-up database; similar in function to DNS. Interpreted in the Sun PI suite.

ZIP Zone Information Protocol. Used in AppleTalk to maintain an internet-wide mapping of networks to zone names. ZIP is used by the Name-Binding Protocol NBP to determine which networks belong to a given zone. Interpreted the AppleTalk PI suite.

Zone In AppleTalk networks, a set of one or more networks within an internet, such that no network is a member of more than one zone.

Appendix B. Bibliography

General

Martin, James. *Local Area Networks.* The Arben Group, 1989.

Meijer, Anton and Paul Peeters. *Computer Network Architectures.* Rockville, Maryland: Computer Science Press, 1982.

Stallings, William. *Local Networks.* Macmillan, 1990.

Tannenbaum, Andrew S. *Computer Networks.* 2d ed. Englewood Cliffs: New Jersey, 1989.

Networks

ARCNET

ARCNET Designer's Handbook. Datapoint Corporation, publication number 61610-01.

Attached Resource Compute: Simplified User's Guide. Datapoint Corporation, document number 50298, revision number 1.

Concepts of ARC Local Networking. Datapoint Corporation, document number 50694.

Herman, Mort. "LAN controller regulates token-passing traffic," *Electronic Design,* December 22, 1983, 139-144.

Local Area Network Controller LANC. Standard Microsystems Corporation, document number 4/83-2.5M.

Murphy, John A. "Token-passing protocol boosts throughput in local networks," *Electronics,* September 8, 1982.

Shustek, Leonard J. "A Client-Server Protocol for Local Area Networks." *Systems and Software* (March 1984): 127-131.

Ethernet and StarLAN

IEEE Standards for Local Area Networks: *Logical Link Control.* ANSI/IEEE Std 802.2-1985 (ISO/DIS 8802/2). IEEE publication number SH09712. Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017.

IEEE Standards for Local Area Networks: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Access Method and Physical Layer Specifications.* ANSI/IEEE Std 802.3-1985 (ISO/DIS 8802/3). IEEE publication number SH09738. Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017.

The Ethernet: A Local Area Network. Data Link Layer and Physical Layer Specifications. ("The blue book.") Issued jointly by Digital Equipment Corporation, Maynard, MA, Intel Corporation, Santa Clara, CA, and Xerox Corporation, Stamford, CT. Version 2.0, November 1982. Available from: Hillary Cornell, Xerox Systems Institute, 475 Oakmead Parkway, Sunnyvale, CA 94086. (408) 737-4652.

Token Ring

Haugdahl, J. Scott. *Inside the Token Ring.* Architecture Technology Corporation, 1986. P.O. Box 24344, Minneapolis MN 55424.

IEEE Standards for Local Area Networks: Logical Link Control. ANSI/IEEE Std 802.2-1985 (ISO/DIS 8802/2). IEEE publication number SH09712. Institute of Electrical and Electronics Engineers, 345 East 47th Street, New York, NY 10017.

IEEE Standards for Local Area Networks: Token Ring Access Method. ANSI/IEEE Std 802.5-1985 (ISO/DP 8802/5), IEEE publication number SH09944.

TMS380 Adapter Chipset User's Guide. Texas Instruments Incorporated, publication number SPWU001.

Token Ring Network Architecture Reference. IBM Corporation, publication number 6165877.

Token Ring Network PC Adapter Technical Reference. IBM Corporation, publication number 69X7713.

IBM PC Network

Berry, Paul. *Operating the IBM PC Network.* Berkeley, CA: SYBEX, Inc., 1986.

Cooper, Edward. *Broadband Network Technology: An Overview for the Data and Telecommunications Industry.* Englewood Cliffs, New Jersey: Prentice-Hall, 1984.

Cunningham, John E. *Cable Television,* Second Edition. Indianapolis: Howard W. Sams & Co., 1980.

Hewlett Packard. *Cable Television System Measurements Handbook.* Santa Rosa, CA: Hewlett Packard Co,, 1977.

National Cable Television Association. *Standards of Good Engineering Practices for Measurement on Cable Television Systems, Distribution System.* Washington, D.C.: NCTA, 1977.

PC Network Technical Reference. IBM Corporation, publication number 6322916.

Shrock, Clifford B. *No Loose Ends: The Tektronix Proof-of-Purchase Program for CATV.* Tektronix Application Note, 1973.

Simons, Ken. *Technical Handbook for CATV Systems,* 3rd Edition. Hatboro, PA: General Instrument Corporation, Jerrold Division, 1968.

LocalTalk

Sidhu, Gursharan S., Richard F. Andrews and Alan B. Oppenheimer. *Inside AppleTalk.* Addison-Wesley Publishing Company, 1989.

Synchronous

Data Communication Networks Interfaces: Recommendations X.20—X.32, Red Book, Volume VIII-Fascicle VIII.3. Geneva: International Telecommunications Union-CCITT, 1985.

Protocols

IBM

Advanced Program-to-Program Communication for the IBM Personal Computer. Programming Guide. IBM Corporation, publication number 61X3842.

Haugdahl, J. Scott. *Inside NetBIOS.* Architecture Technology Corporation, 1986. P.O. Box 24344, Minneapolis MN 55424.

Systems Network Architecture Reference Summary. IBM Corporation, publication number GA27-3136.

Novell NetWare

Sheldon, Tom. *Novell NetWare: The Complete Reference.* Berkeley, California: McGraw-Hill, 1989.

XNS

Internet Transport Protocols. Xerox Systems Integration Standard X.S.I.S. 028112, December, 1981.

TCP/IP

Cerf, V.G. and R.E. Kahn. "A Protocol for Packet Network Interconnection." *IEEE Trans. Commun.* COM-22:637—648 (May, 1974).

Comer, Douglas E. *Internetworking With TCP/IP: Principles, protocols, and Architecture.* Englewood Cliffs, New Jersey: Prentice-Hall, 1988.

DDN Protocol Handbook.
Vol. 1: DOD Military Standard Protocols. NIC-5004.
Vol. 2: DARPA Internet Protocols. NIC-5005.
Vol. 3: Supplement. NIC-5006.
US Defense Communications Agency, December, 1985. Available from: DDN Network Information Center, DDN Network Information Center, SRI International, Room EJ291, 333 Ravenswood Avenue, Menlo Park, CA 94025 (800) 235-3155; (415) 859-3695. NIC@SRI-NIC.ARPA;
or from Defense Technical Information Center, Cameron Station, Alexandria, VA 22314 (202) 274-7633.

TCP/IP Interoperability Conference. Transcript/notes published by Advanced Computing Environments, 21370 Vai Avenue, Cupertino, CA 95014.

OSI

Day, J.D. and H. Zimmerman. "The OSI Reference Model." *Proceedings of the IEEE* 71 (1983): 1334-1340.

Henshall, J. and A. Shaw. *OSI Explained. End to End Computer Communication Standards.* Chichester, England: Ellis Horwood, 1988.

Linington, P.F. "Fundamentals of the Layer Service Definitions and Protocol Specifications." *Proceedings of the IEEE* 71 (1983): 1341-1345.

Rose, Marshall T. *The Open Book: A Practical Perspective on OSI.* Englewood Cliffs, New Jersey, 1990.

Zimmerman, H. "OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection." *IEEE Trans. Commun.* COM-28:425—432 (April 1980).

DECnet

Malamud, Carl. *DEC Networks and Architectures.* New York: McGraw-Hill Book Company, 1989.

AppleTalk

Sidhu, Gursharan S., Richard F. Andrews and Alan B. Oppenheimer. *Inside AppleTalk.* Addison-Wesley Publishing Company, 1989.

X Windows

Scheifler, Robert, James Gettys, and Ron Newman. *X Window System, Library, and Protocol Reference.* Digital Press, 1988.

X.25

Deasington, R.J. *X.25 Explained: Protocols for Packet Switched Networks.* 2d ed. Chichester, England: Ellis Horwood, 1988.

Dhas, C.R. and V.K. Konangi. "X.25: An Interface to Public Packet Networks." *IEEE Commun. Magazine* IT-22 (1976): 118-125.

The X.25 Protocol and Seven Other Key Protocols. Belmont, California: Lifetime Learning Publications.

SNA

IBM Systems Network Architecture Formats. IBM Corporation, publication number GA27-3136-10.

Index

Index

Network General

Index

Index

Network General Corporation
4200 Bohannon Drive
Menlo Park, CA 94025